Escuela Politécnica Superior

Transformación de emociones mediante redes generativas antagónicas

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
David Azuar Alonso

Tutor/es:
Miguel Ángel Cazorla Quevedo
Francisco Gómez Donoso

Junio 2019

Universitat d'Alacant
Universidad de Alicante

Transformación de emociones mediante redes generativas antagónicas

Autor

David Azuar Alonso

Tutores

Miguel Ángel Cazorla Quevedo

Departamento de Ciencia de la Computación e Inteligencia Artificial

Francisco Gómez Donoso

Departamento de Ciencia de la Computación e Inteligencia Artificial

Grado en ingeniería informática

Escuela Politécnica Superior

Alicante, Junio 2019

Preámbulo

En este proyecto se va a realizar un trabajo de investigación sobre las redes generativas antagónicas para entender cómo funcionan y poder entrenar la arquitectura CycleGAN utilizando datasets del campo de las expresiones faciales para conseguir transformar las emoción presente en un rostro en una imagen a otras diferentes. Conseguir esta transformación puede abrir la puerta a nuevas técnicas de data augmentation en datasets del campo del reconocimiento de expresiones que ayuden a paliar el desequilibrio entre las diferentes emociones presente en la mayoría de ellos o servir para generar datos para una aplicación que ayude a las personas con problemas para reconocer emociones.

Agradecimientos

Me gustaría aprovechar este espacio en el trabajo para acordarme de todas las personas que me han ayudado a llevarlo a buen puerto y me han soportado durante todo el proceso, pese a que en ocasiones lo haya puesto más difícil de la cuenta.

Me gustaría acordarme de mis tutores, Miguel y Fran, que me han ayudado en absolutamente todo lo que he necesitado y me han dado la oportunidad de trabajar en su departamento. También dar las gracias a las personas de este departamento que me han descubierto una forma diferente de hacer las cosas. Ha sido un placer trabajar con vosotros.

Quiero también agradecer a mis compañeros de carrera que durante este último año nos hayamos apoyado mucho y entre todos nos hayamos ayudado en cualquier cosa. Sin ellos este año se hubiese afrontado de una manera muy diferente.

Por último no puedo dejar de acordarme de mi familia, y de mis amigos y amigas que son también como otra familia. Sé que durante este año no he sido la persona más fácil pero habéis seguido estando ahí.

A todos vosotros, muchas gracias.

Mirar demasiado puede impedirte ver.

Patrick Rothfuss.

Índice general

1. Introducción **1**
1.1. Objetivos 3

2. Marco teórico **5**
2.1. Conceptos previos 5
2.1.1. Machine Learning 5
2.1.2. Redes neuronales 6
2.1.3. Redes convolucionales 7
2.2. Redes GAN 11
2.2.1. CycleGAN 14
2.2.2. Otras redes valoradas 18

3. Metodología **23**
3.1. Aproximación 23
3.2. Herramientas utilizadas 25

4. Desarrollo **29**
4.1. Dataset 29
4.1.1. Conjuntos encontrados 29
4.1.2. Análisis de los datasets encontrados 34
4.1.3. Balanceado de dataset 36
4.1.4. Procesado de dataset 37
4.2. Selección de implementación de CycleGAN 39
4.2.1. Criterios de selección 39
4.2.2. Implementaciones encontradas 40

5. Experimentación **45**
5.1. Primer entrenamiento 45
5.2. Entrenamiento con más imágenes 48

5.3. Imágenes sin procesar contra imágenes procesadas 50
5.4. Entrenamientos para otras emociones 52
5.5. Entrenamiento con más épocas 55
5.6. Inferencia en tiempo real 59

6. Conclusiones **61**

Bibliografía **65**

A. Anexo I: Preparación de entorno y entrenamiento de la CycleGAN **67**
A.1. Instalación de dependencias 67
A.2. Entrenamiento 69

Índice de figuras

2.1. Esquema de la estructura de una red neuronal 6
2.2. Los coches autónomos también utilizan NN para la navegación 7
2.3. Estructura de una CNN 8
2.4. Funcionamiento de un filtro convolucional 8
2.5. Funcionamiento del *pooling* 9
2.6. Ejemplo de deteccion de objetos 9
2.7. Imagen de retinopatía del dataset Messidor-2 10
2.8. Ejemplo de detección de caras 10
2.9. Estructura encoder-decoder 11
2.10. Ejemplos de segmentación con FCN 11
2.11. Estas personas no existen, han sido generadas por una GAN de nvidia[1] . 12
2.12. Estructura general de una GAN 13
2.13. Esquema de la CycleGAN 14
2.14. Ejemplo de *cycle consistency* 15
2.15. Estructura del generador de la CycleGAN 16
2.16. Estructura del discriminador de la CycleGAN 17
2.17. CycleGAN aplicada a transferencia de estilo artístico 18
2.18. CycleGAN transformando entre caballos y zebras 18
2.19. Arquitectura de CGAN 19
2.20. Arquitectura de IcGAN[2] 20
2.21. IcGAN aplicada a CelebA[2] 20

3.1. Esquema de utilización de la CycleGAN para transformar emociones . . . 23

4.1. Muestras de Oulu-CASIA dataset 30
4.2. Muestras de Cohn-Kanade AU-Coded Facial Expression Database 30
4.3. Muestras de FER2013 dataset 31
4.4. Muestras de EmotioNet dataset 32
4.5. Muestras de AffectNet dataset 33

4.6. Detección de cara 37
4.7. Proceso de rotación 38
4.8. Recorte final 38
4.9. Muestras del entrenamiento de la implementación en Keras 42
4.10. Muestras del entrenamiento de la implementación en Tensorflow 43

5.1. Proceso de reconstrucción 46
5.2. Resultados del entrenamiento con 1000 imágenes para la emoción de felicidad y 200 épocas 47
5.3. Resultados del entrenamiento con 2000 imágenes para la emoción de felicidad y 200 épocas 49
5.4. Resultados del entrenamiento con 2000 imágenes para la emoción de felicidad sin procesar y 200 épocas 51
5.5. Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 200 épocas 53
5.6. Resultados del entrenamiento con 2000 imágenes para la emoción de asco y 200 épocas 54
5.7. Graficas loss 55
5.8. Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 300 épocas 57
5.9. Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 2000 épocas 58
5.10. Captura de inferencia en tiempo real a felicidad 59
5.11. Captura de inferencia en tiempo real a asco 59

1. Introducción

El reconocimiento de emociones es un factor clave en las relaciones humanas. Las personas sabemos detectar las formas que adquieren los rostros de nuestros interlocutores para interpretar su estado emocional o sus intenciones. Entonces, en este sentido, podemos decir que también es una tarea clave que deben dominar los robots sociales y es también una rama de investigación de gran interés para referentes tecnológicos como Google[1], Facebook, Microsoft o Apple[2] que en los últimos años han invertido grandes cantidades de dinero en hacerse con empresas líderes en este campo con el objetivo de desarrollar estas tecnologías para, por ejemplo, hacer que el diálogo con ordenadores sea mucho más significativo.

Las aproximaciones convencionales para afrontar este problema se basaban en tres pasos diferenciados en los que es localizaba la región de la cara en una imagen, se extraían las características de los componentes faciales y a través de estas características se llevaba a cabo una clasificación. Sin embargo, con el auge de posibilidades que ofrece la inteligencia artificial en la actualidad, estas aproximaciones tradicionales se han visto reemplazadas por alternativas basadas en *Deep Learning* que agrupan el proceso anterior y que requieren de grandes cantidades de datos etiquetados para ser entrenadas[3]. Sin embargo, los conjuntos de datos existentes tienden a padecer de diferentes problemáticas: las imágenes que lo conforman son insuficientes, son locales a un tipo de población, los ejemplos son poco naturales o existe un gran desbalanceo entre la cantidad de ejemplos de una y otra clase. Debido a esto, incluso los sistemas más sofisticados, funcionan muy por debajo del nivel de acierto de un humano.

Para solucionar esto, en este trabajo se propone la generación automática de imágenes de caras con diferentes expresiones a partir de caras con emoción neutra, con la intención de generar sin esfuerzo grandes cantidades de datos que permiten a la postre entrenar

[1] https://www.kairos.com/blog/face-recognition-kairos-vs-microsoft-vs-google-vs-amazon-vs-opencv
[2] https://www.wired.com/2016/01/apple-buys-ai-startup-that-reads-emotions-in-faces/

sistemas de reconocimientos de emociones más precisos. En concreto, nuestro método usa redes generativas antagónicas o redes GAN, que son un tipo de redes neuronales centradas en la generación de datos y que están en auge en la actualidad. La razón por la cual se toman imágenes neutras es porque representan el mayor número de ejemplos en los *datasets* del estado del arte, llegando en algunos casos a proporciones de 20 a 1 en relación a las emociones menos representadas como asco o sorpresa. Por otro lado, no es posible sobreactuar dicha expresión y es por eso que representa el dominio más estable de entre todas las emociones.

El resto del documento se estructura de la siguiente manera. En el capítulo 2 se realiza una introducción teórica a las redes generativas antagónicas así como diferentes arquitecturas y la CycleGAN, que será la que finalmente usemos. En el capítulo 3, se explica el planteamiento que se ha seguido en el desarrollo y se explican las herramientas y tecnologías que se han utilizado. El 4 se centra en el proceso de desarrollo y en él se detalla cómo se ha llevado a cabo el análisis y la selección de dataset, el sistema de procesado del mismo así como la selección de implementación de CycleGAN a utilizar, desarrollando los problemas encontrados y las soluciones propuestas. El capítulo 5 se dedica a agrupar todo el proceso de experimentación y mostrar los resultados obtenidos y finalmente en el capítulo 6 se hace una valoración del proyecto y se proponen posibles trabajos futuros a abordar.

1.1. Objetivos

El objetivo principal de este proyecto es entrenar diferentes modelos de la red CycleGAN que consigan transformar imágenes de rostros con expresión neutra a otras que presenten esas mismas caras pero con otra emoción.

Para conseguir este objetivo se van a llevar a cabo las siguientes tareas o sub-objetivos:

- Investigar sobre las redes generativas antagónicas para entender cómo funcionan.
- Buscar y analizar los datasets existentes con imágenes de caras etiquetadas con la emoción que representan.
- Encontrar una implementación de la CycleGAN que funcione acorde a nuestras necesidades.
- Entrenar modelos de la CycleGAN para transformar a diferentes emociones.
- Implementar un procesado de las imágenes previo al entrenamiento para mejorar los resultados finales
- Mejorar las configuraciones de entrenamiento para conseguir mejores resultados.

2. Marco teórico

En este capítulo se va a realizar una introducción teórica a las redes generativas antagónicas o redes GAN que van a ser, como se ha comentado anteriormente en el documento, las responsables de llevar a cabo las tareas de modificar las expresiones de las imágenes de rostros.

Primero se van a explicar unos conceptos previos y a continuación se entrará a explicar en detalle este tipo de redes así como diferentes arquitecturas que se han valorado para utilizar.

2.1. Conceptos previos

El primer ámbito a comentar para situar nuestro trabajo a nivel teórico es el del campo del *Machine Learning* o aprendizaje automático, puesto que es en este en el que se engloban las sucesivas tecnologías que darán finalmente lugar al tipo de redes que vamos a utilizar.

2.1.1. Machine Learning

El *Machine Learning* es una rama de la inteligencia artificial que tiene como objetivo conseguir llevar a cabo una tarea determinada sin para ello utilizar instrucciones explícitas[1], basándose en la inferencia y los patrones en su lugar. Se utilizan conjuntos de datos de ejemplo para entrenar un modelo matemático que, una vez entrenado, deberá ser capaz de hacer predicciones correctas para nuevas entradas. Tiene una gran cantidad de aplicaciones que van desde el reconocimiento de objetos en imágenes hasta

[1] https://en.wikipedia.org/wiki/Machine_learning

el filtrado de correos electrónicos. Se han desarrollado a lo largo del tiempo multitud de aproximaciones como las *Support Vector Machines* o SVM, los algoritmos genéticos o las redes neuronales, siendo estas últimas en las que estamos interesados puesto que son utilizadas en la estructura de las redes GAN como veremos más adelante.

2.1.2. Redes neuronales

De cara a llevar acabo el "aprendizaje" que propone el Machine Learning, podemos decir brevemente que las redes neuronales se basan en la interconexión de las llamadas neuronas artificiales entre sí de forma que queden estructuradas en capas[2]. La información de entrada se introduce al sistema a través de la primera capa de neuronas y atraviesa la red siendo sometida a múltiples operaciones de tal forma que los valores que queden en la capa final se correspondan con la salida esperada.

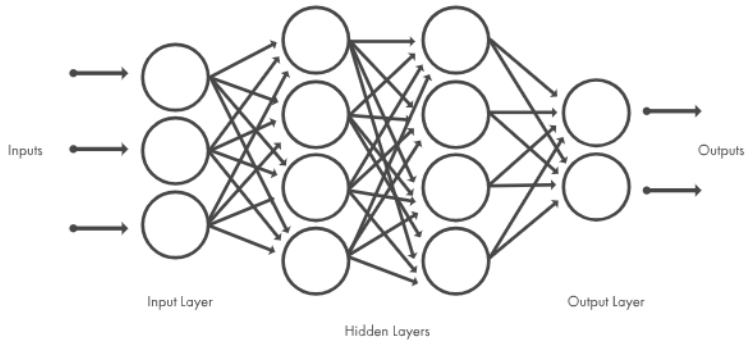


Figura 2.1.: Esquema de la estructura de una red neuronal

Existen infinidad de variantes y modelos que juegan con la estructura de las capas y la forma de interconectar las neuronas y las aplicaciones en las que son utilizadas van desde la clasificación basada en el reconocimiento de patrones, para por ejemplo detectar si un correo electrónico es spam o no; a la aproximación de funciones, que permiten, por ejemplo, que un robot aprenda a qué velocidad y en qué dirección debe moverse dependiendo de los valores que leen sus sensores.

[2]https://es.wikipedia.org/wiki/Red_neuronal_artificial

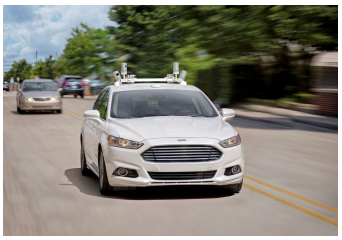


Figura 2.2.: Los coches autónomos también utilizan NN para la navegación

2.1.3. Redes convolucionales

Las redes neuronales convolucionales o CNN son una derivación de las redes neuronales tradicionales que introducen nuevos elementos a su estructura que le permiten especializarse en las aplicaciones que utilizan imágenes como fuente de ejemplos para el aprendizaje. Esta condición les acerca más al ámbito del trabajo y les hace formar parte del tipo de redes que finalmente utilizaremos como veremos en el siguiente apartado y es por eso que vale la pena conocer su funcionamiento.

Funcionamiento

Los nuevos elementos que estas redes introducen son principalmente las capas de convolución y las de *pooling*. Estas nuevas capas se sitúan en la entrada de la red y permiten automatizar el proceso de extracción de características, eliminando la necesidad de extraer las mismas a mano previamente utilizando conocimientos sobre el campo de los datos y esfuerzo humano. Por ejemplo, para entrenar una red tradicional para detectar las líneas de la carretera sería recomendable aplicar a las imágenes un filtro para detectar bordes mientras que en el caso de utilizar una red convolucional no sería necesario. Después de esas nuevas capas encontraríamos la estructura de una red tradicional, con capas de neuronas totalmente conectadas que utilizarían las características encontradas para finalmente clasificar la imagen.

Las capas de convolución tienen la función de aplicar filtros convolucionales sobre la imagen de entrada para conseguir encontrar las características que mejor nos permiten obtener más precisión en el resultado final. Cada capa se configura para aplicar diferente número de filtros y durante el proceso de entrenamiento y aprendizaje de la red, el valor de estos filtros se modifica como si se tratase de pesos de las neuronas de las redes

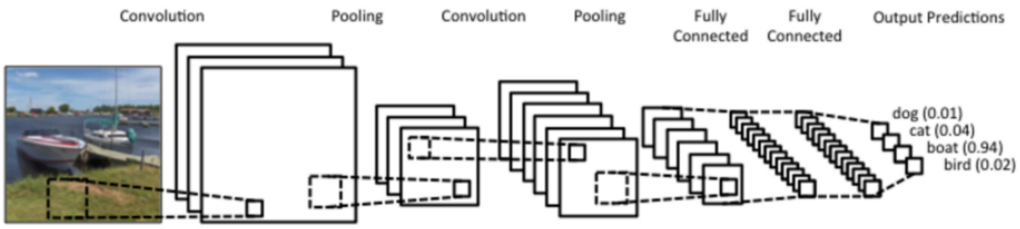


Figura 2.3.: Estructura de una CNN

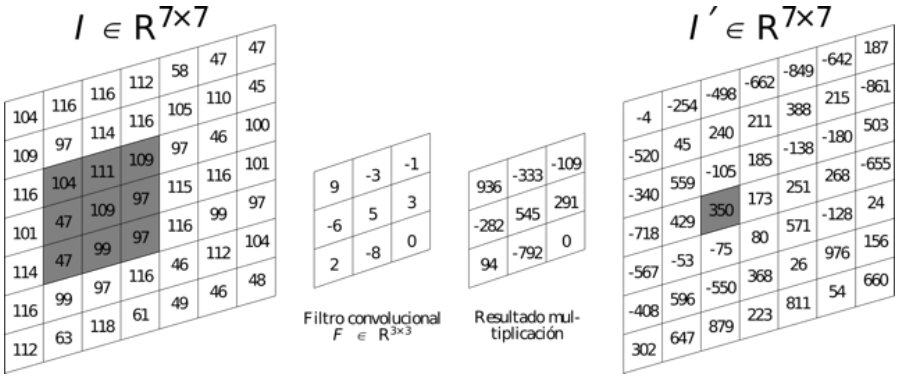


Figura 2.4.: Funcionamiento de un filtro convolucional

tradicionales para finalmente llegar a encontrar los que mejor funcionan.

Las capas de *pooling* por otra parte, son las encargadas de reducir las dimensiones de los datos de cara a intentar sintetizar las particularidades de la imagen de entrada lo máximo posible. Se basan en mapear subconjuntos de píxeles de la entrada en uno para la imagen de salida aplicando diferentes funciones dependiendo de lo que busquemos. Una de las capas más utilizadas es la *MaxPooling2D*, que utiliza cuatro píxeles de la imagen original por cada uno de la final, reduciendo sus dimensiones a la mitad en ambos ejes, y dando al píxel final el máximo valor presente entre los cuatro de la imagen original.

Aplicaciones

Este tipo de redes está consiguiendo alcanzar unos resultados que antes se consideraban inalcanzables en campos como el reconocimiento de objetos en imágenes o el de clasificación, y su proliferación ha supuesto el desbancamiento de otros métodos de ML

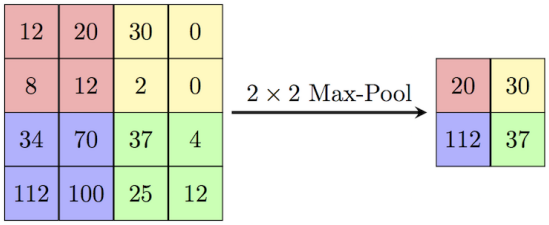


Figura 2.5.: Funcionamiento del *pooling*

tradicionales que se venían utilizando hasta el momento.

Uno de los ejemplos que cabe destacar es el de competiciones como la ImageNet Large Scale Visual Recognition Challenge[3], que enfrentan anualmente a múltiples equipos para ver qué alternativa proporciona mejores resultados tanto en clasificación de imágenes como en detección de objetos sobre su extenso dataset multiclase [4] y que desde la llegada de las CNN han visto como los resultados son drásticamente mejores.

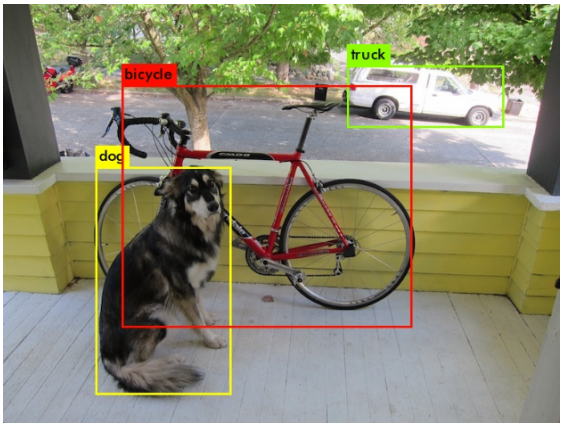


Figura 2.6.: Ejemplo de deteccion de objetos

Otras de las aplicaciones de relevancia de este tipo de redes son las de carácter médico. Se han utilizado redes convolucionales para que, por ejemplo, entrenando la red con imágenes médicas de sujetos que padecen o no retinopatía[4] y de su grado, esta sea capaz de diagnosticar si un nuevo paciente sufre la enfermedad o no, alcanzando cotas de precisión similares a las de oftalmólogos expertos pero en un tiempo mucho menor [5].

Finalmente, me gustaría mencionar también la utilización de las CNN en el campo

[3]http://image-net.org/challenges/LSVRC/2016/index
[4]https://www.macula-retina.es/inteligencia-artificial-para-detectar-retinopatia-diabetica/

Figura 2.7.: Imagen de retinopatía del dataset Messidor-2

de la detección de caras y el reconocimiento facial. Más adelante, cuando se explique el procesado de las imágenes previo al entrenamiento de la red, veremos un ejemplo práctico del primer punto.

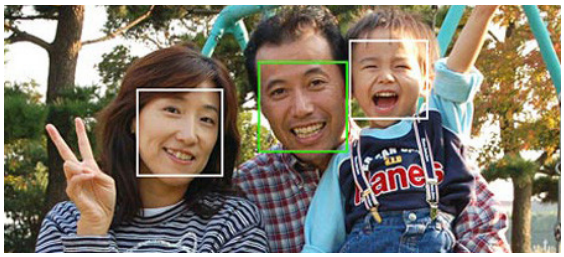


Figura 2.8.: Ejemplo de detección de caras

Redes totalmente convolucionales y deconvolucionales

Por último, antes de introducir las redes GAN es importante conocer estas últimas derivaciones de las CNN puesto que serán los tipos de redes que mayoritariamente conformaran la estructura de las redes GAN.

Estos tipos de redes también son conocidos como encoders y decoders convolucionales y están formados íntegramente por las capas características de las redes CNN que se han visto en el punto anterior. En el ámbito del tratamiento de imágenes, el encoder tiene la función de "comprimir" la imagen de entrada reduciendo su dimensionalidad a un vector a partir del cual el decoder pueda reconstruir la imagen.

Este tipo de redes suelen ser entrenadas con imágenes emparejadas de dos dominios con el objetivo de que una vez entrenada, introduciendo una imagen del primer dominio en la red, obtengamos a la salida su equivalente en el segundo dominio.

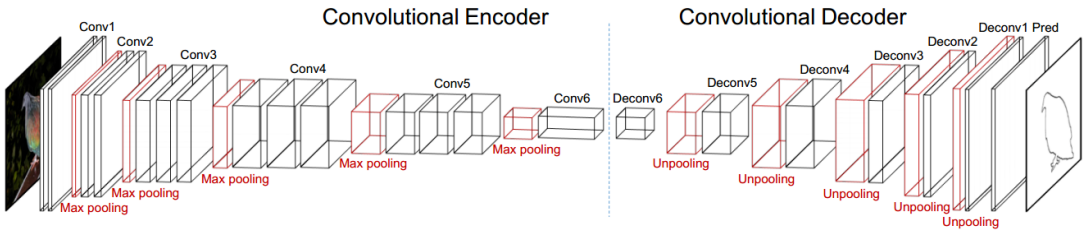


Figura 2.9.: Estructura encoder-decoder

Uno de los casos en los que se utiliza es en el de la segmentación de imágenes[6]. Se entrena utilizando imágenes reales y sus contrapartidas segmentadas por otro tipo de algoritmo o a mano. Los resultados obtenidos son muy buenos como podemos ver en la figura 2.10.

(a) (b)

Figura 2.10.: Ejemplos de segmentación con FCN

2.2. Redes GAN

Finalmente, después de entrar en contexto con la explicación de las tecnologías previas podemos entrar a comentar las redes GAN o redes generativas antagónicas [7] que serán las que finalmente utilizaremos en este trabajo.

Este tipo de redes son capaces de generar imágenes totalmente nuevas del dominio de las imágenes con el que ha sido entrenada. Por ejemplo, si entrenáramos nuestra red con miles de fotos de rostros de personas, una vez hubiera concluido el entrenamiento esta sería capaz de generar rostros de personas que nunca han existido pero que a nuestros

ojos parecen completamente reales. En la figura 2.11 se puede ver un ejemplo de esta aplicación.

(a) (b) (c)

Figura 2.11.: Estas personas no existen, han sido generadas por una GAN de nvidia[1]

De cara a conseguir este objetivo, las redes GAN están compuestas por otras dos redes, una red generadora y otra discriminadora, que son enfrentadas mutuamente. La red generadora es la encargada de generar nuevas imágenes que la discriminadora valorará para juzgar si la imagen que le llega ha sido generada o si de lo contrario es una imagen real del dominio de las imágenes de entrenamiento. Se dice que ambas redes están enfrentadas puesto que la red discriminadora está intentando aprender a distinguir entre las imágenes reales y las provenientes del generador, mientras que la generadora pretende que sean indistinguibles. En el ejemplo anterior de la red que intenta generar rostros de personas el generador al comienzo del entrenamiento generaría imágenes con mucho ruido que el discriminador no tendría ningún problema en distinguir como no caras. Sin embargo, el afán de la red generadora de maximizar el error del discriminador haría que conforme progresa el entrenamiento, se vayan generando en las imágenes las diferentes formas y características que conforman un rostro y que, finalmente, conseguirían que dichas imágenes generadas no sean distinguibles de las reales.

En el modelo original, la red discriminadora era una convolucional y la generadora era una deconvolucional (figura 2.12), pero desde que apareció se han desarrollado multitud de derivaciones que alteran esta estructura de cara a conseguir llevar las GANs a diferentes aplicaciones.

Antes de entrar a ver el funcionamiento de modelos concretos de este tipo de redes que se han valorado de cara al desarrollo de trabajo podemos sintetizar el funcionamiento de las GAN visto anteriormente.

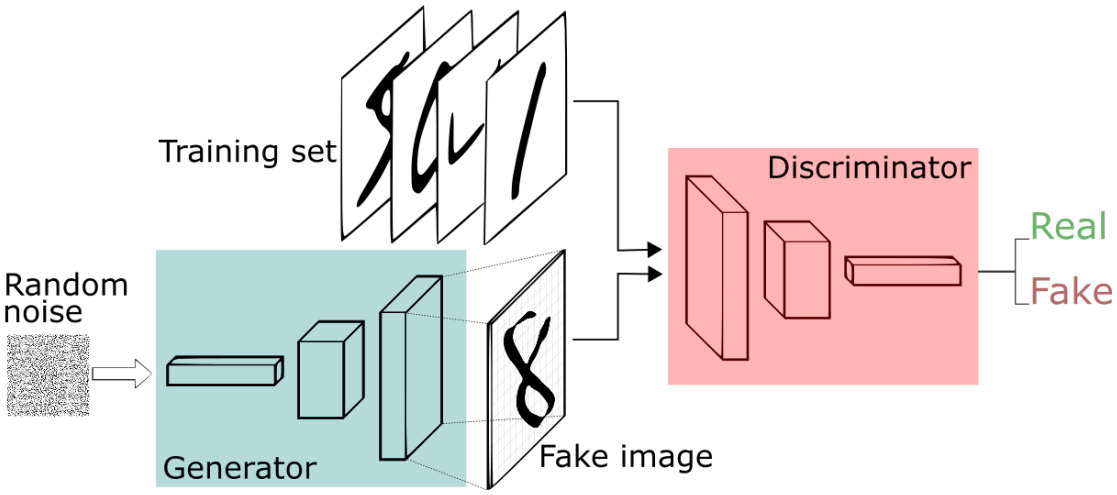


Figura 2.12.: Estructura general de una GAN

Esta es la secuencia de pasos que sigue una red GAN durante su entrenamiento:

- El generador genera una imagen a partir de un vector de ruido.
- La imagen generada se pasa por el discriminador junto a imágenes reales del conjunto de entrenamiento.
- La red discriminadora devuelve probabilidades de como de real piensa que es cada imagen.
- La red discriminadora actualiza sus pesos tratando de minimizar el error del paso anterior.
- La red generadora se actualiza para tratar de, por el contrario, maximizar el *loss* o error de la red discriminadora.

2.2.1. CycleGAN

Habiendo introducido las GAN, podemos entrar a explicar la CycleGAN [8] que es la arquitectura que finalmente escogeremos para llevar a cabo nuestro trabajo. Hablaremos de su funcionamiento, estructura y de ejemplos de aplicaciones en las que se ha utilizado.

La CycleGAN es una implementación de una red GAN que tiene como objetivo traducir una imagen de un dominio X a un dominio objetivo Y sin utilizar en el entrenamiento conjuntos de imágenes emparejadas como sí se utilizan en el caso por ejemplo de las redes totalmente convolucionales que hemos visto anteriormente. De cara a tratar de solventar la carencia que esta condición supone, la red introduce otra transformación de la imagen generada al dominio de origen, generando el *cycle consistency loss* al comparar la imagen original con esta reconstrucción a partir de la imagen traducida al otro dominio. Este proceso permite a la red que, teniendo en cuenta este *loss* durante el entrenamiento, pueda evitar la pérdida de identidad de la imagen original aún trasladada al segundo dominio.

Arquitectura y funcionamiento

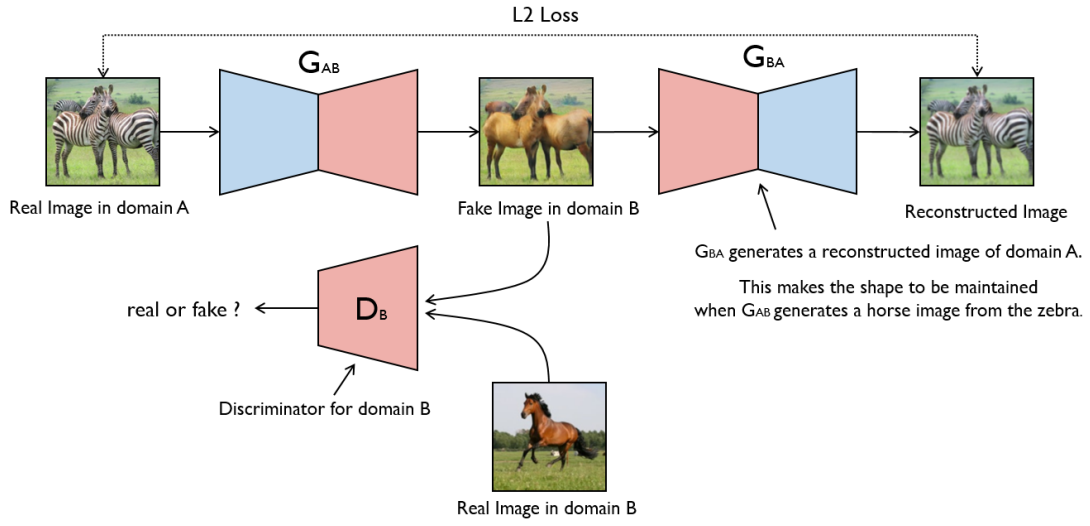


Figura 2.13.: Esquema de la CycleGAN

De esta manera la red esta formada por dos generadores y dos discriminadores. El

primer generador G convierte imágenes del dominio X al dominio Y, mientras que el segundo, F, convierte imágenes de Y a X.

Cada generador tiene su correspondiente discriminador que valorará si las imágenes que le llegan son generadas o reales, proporcionando la información para el entrenamiento tradicional de las GAN que hemos visto en apartados anteriores.

Sin embargo, dicho entrenamiento no es suficiente para "traducir" imágenes entre dos dominios puesto que no garantiza que la imagen traducida esté relacionada con la original. Por ejemplo, uno de los conjuntos con los que se ha entrenado esta red es el de imágenes de paisajes e imágenes de cuadros de un artista para tratar de obtener una imagen que recree el paisaje pero con el estilo del artista en cuestión, como veremos más adelante; si intentáramos conseguir este objetivo utilizando el entrenamiento tradicional, la imagen obtenida perdería la identidad del paisaje puesto que el generador tendería a transformar la imagen original a una imagen que el discriminador identificase como real, consiguiendo en la mayoría de los casos que dicha imagen final perdiese por completo cualquier relación con la original: si el generador genera una imagen de un cuadro que representa fielmente el estilo del artista, aunque no tenga nada que ver con la imagen original el discriminador no sería capaz de distinguirla de una real cumpliendo el objetivo del entrenamiento sin haber conseguido la traducción.

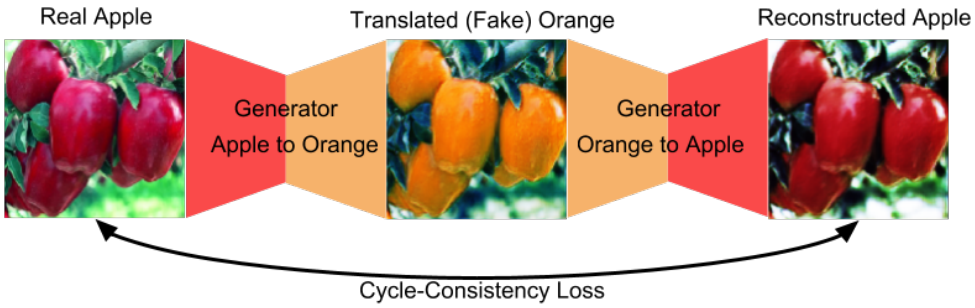


Figura 2.14.: Ejemplo de *cycle consistency*

De cara a paliar este problema se introduce el *cycle consistency loss* (figura 2.14) que hemos introducido anteriormente y que se basa en utilizar los dos generadores para convertir de nuevo la imagen generada mediante el primer generador a través del segundo y que el resultado final se parezca lo máximo posible a la imagen original. De esta manera para dos imágenes x e y de respectivos dominios se trata de conseguir que $F(G(x)) \approx x$ y $G(F(y)) \approx y$.

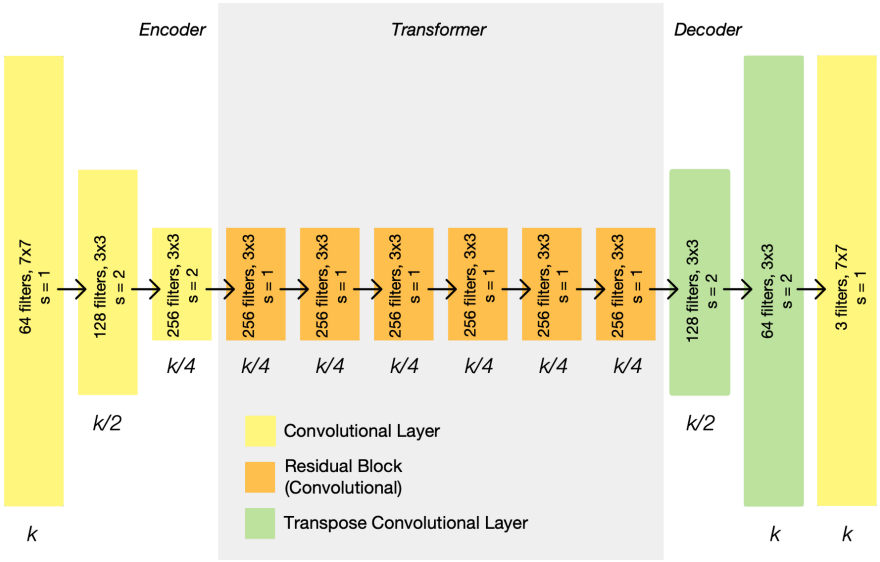


Figura 2.15.: Estructura del generador de la CycleGAN

Por lo que respecta a la estructura de estos generadores y discriminadores[5], cabe destacar que el generador pasa a ser una red totalmente convolucional debido de nuevo a partir de una imagen previa a diferencia de las redes originales que parten de ruido. Esta red totalmente convolucional esta formada por 3 partes diferentes: el encoder, la parte transformadora y el decoder como podemos ver en la figura 2.15.

En cuanto a los discriminadores, la CycleGAN utiliza la estructura del discriminador de la PatchGAN (figura 2.16) que es una red totalmente convolucional que en lugar de valorar la imagen a nivel completo y evaluar si es real o no, proporciona el resultado de esa evaluación para una serie de "parches" de la imagen de entrada.

[5] https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d

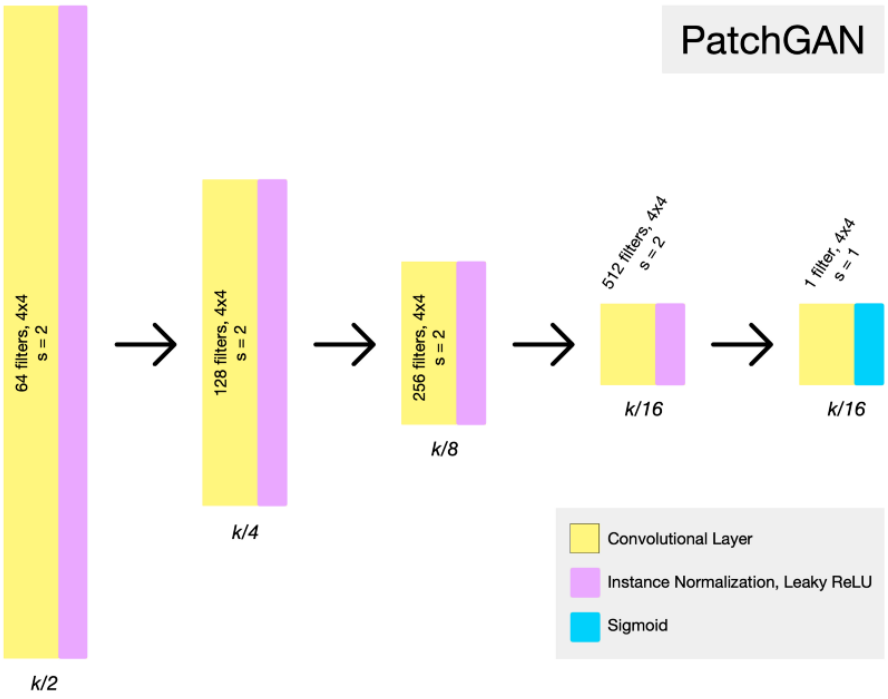


Figura 2.16.: Estructura del discriminador de la CycleGAN

Aplicaciones

La CycleGAN se ha utilizado en una gran cantidad de ámbitos con resultados realmente sorprendentes.

Los autores de la red presentan sus resultados de transferencia de dominio utilizando modelos entrenados con multitud de datasets [9]. En relación a la transferencia de estilo destacan ejemplos como el que se ha mencionado en el apartado anterior relacionados con recrear imágenes de paisajes existentes con estilos de diferentes artistas. En este caso el dominio X sería el de imágenes reales de paisajes mientras que el dominio Y seria el conjunto de cuadros de un determinado artista.

También uno de los datasets a los que más recurren para enseñar cómo funciona su red en el ámbito de la transfiguración de objetos es el de caballos y cebras. El primer dominio X en este caso sería el de los caballos y el objetivo Y sería el de las cebras.

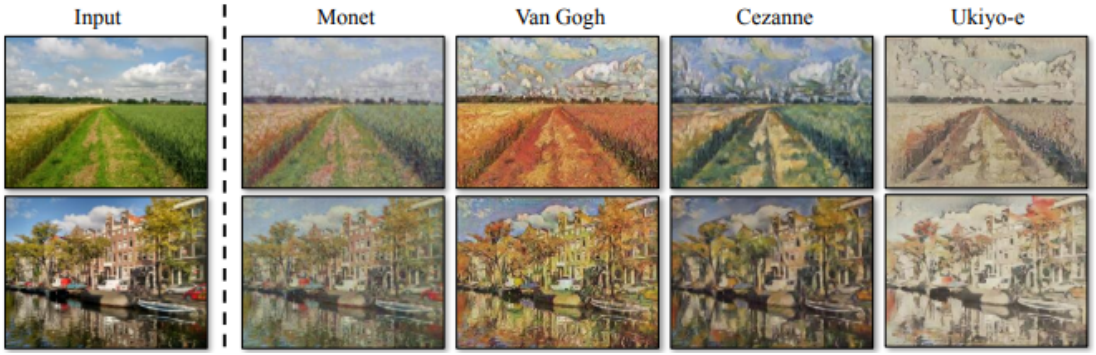


Figura 2.17.: CycleGAN aplicada a transferencia de estilo artístico

(a) (b)

Figura 2.18.: CycleGAN transformando entre caballos y zebras

2.2.2. Otras redes valoradas

A continuación se explicará brevemente el funcionamiento de otras redes que se valoraron para utilizar en el proyecto pero que se descartaron en favor de la CycleGAN.

CGAN

La CGAN o *Conditional GAN* [10] es una extensión de las redes GAN que añade una información extra $\boldsymbol{y}$ al generador y al discriminador para convertir la red en un modelo condicional. Esta información auxiliar suele corresponderse con etiquetas de clase pero se puede introducir otro tipo de datos.

De cara a entender su funcionamiento es mejor poner un ejemplo[6]. Imaginemos que queremos entrenar un modelo para la generación de imágenes de dígitos utilizando el

[6] https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

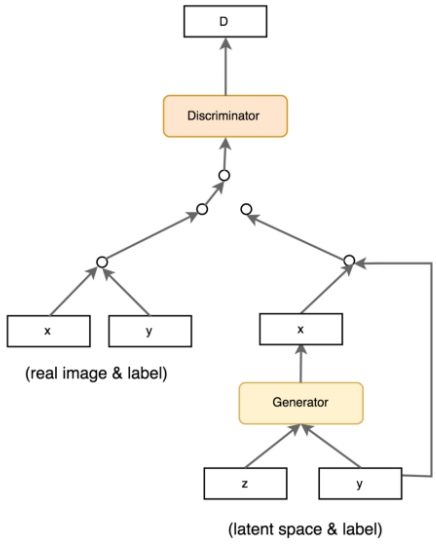


Figura 2.19.: Arquitectura de CGAN

dataset MNIST. La información $\boldsymbol{y}$ sería el propio número que queremos generar que podríamos codificar en un vector. Por ejemplo, si quisiéramos generar un 2, el vector $\boldsymbol{y}$ sería (0,0,1,0,0,0,0,0,0,0). Proporcionando esta información al generador y al discriminador durante el entrenamiento y usando también esa información extra de las imágenes que utilizamos para entrenar, conseguimos poder generar una vez entrenado y con un solo modelo todos los dígitos. Para poder llevar a cabo la tarea anterior en una GAN tradicional, deberíamos entrenar un modelo por cada dígito.

Otro ejemplo en el que se ha utilizado es junto al dataset CelebA, que está compuesto por una gran cantidad de imágenes de rostros de famosos con etiquetas que indican características de la imagen como por ejemplo si la persona lleva gafas de sol, es rubia o si está sonriendo[11]. En este caso, la información extra $\boldsymbol{y}$ sería ese vector de características. Al entrenar la red con estos datos, el modelo final nos permitiría generar imágenes con las características que quisiéramos: mujer rubia y sonriendo, hombre moreno con gafas de sol...

IcGAN

La *Invertible Conditional GAN* [12] es una derivación de la CGAN que tiene como objetivo permitir la modificación de características deliberada presente en dicha arqui-

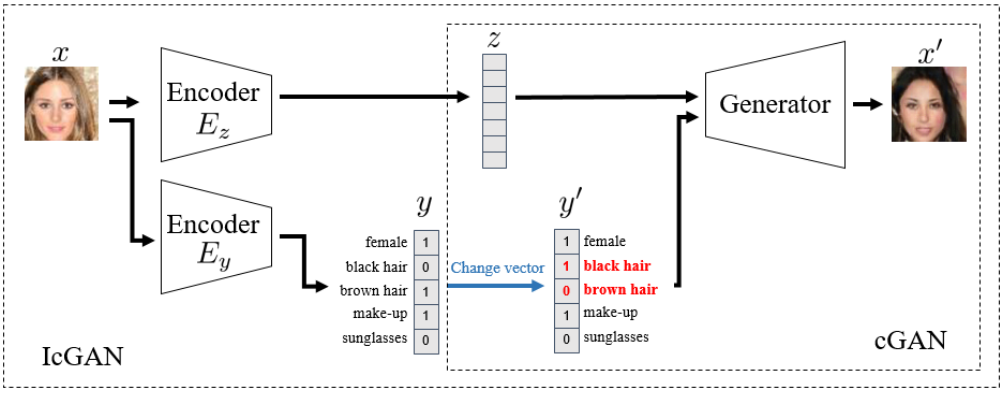


Figura 2.20.: Arquitectura de IcGAN[2]

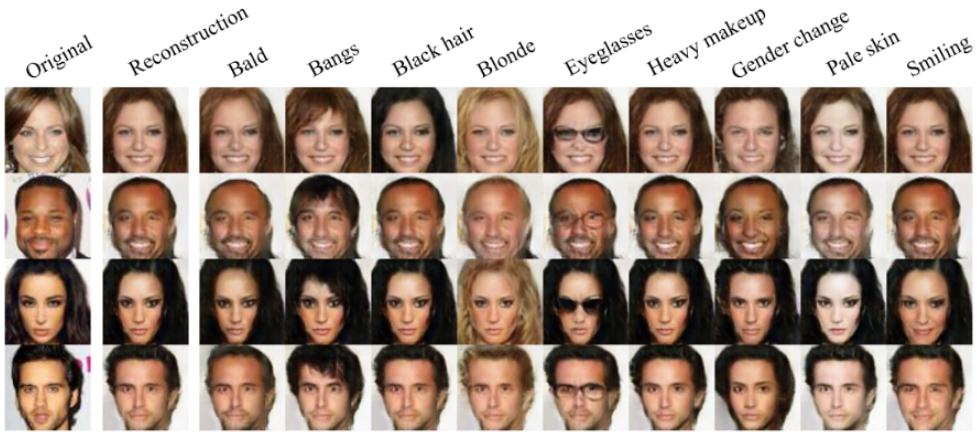


Figura 2.21.: IcGAN aplicada a CelebA[2]

tectura pero sobre imágenes existentes. Para conseguirlo, introduce dos encoders (figura 2.20) que serán entrenados previamente y tendrán como objetivo "comprimir" la imagen de entrada a una representación latente y codificar las características presentes en la imagen respectivamente.

La imagen comprimida se utilizaría en la CGAN como el vector de ruido z, y el vector de características extraído actuaria como la información adicional $\boldsymbol{y}$, pudiendo ser modificado para por ejemplo añadir gafas de sol o cambiar el color del pelo a una persona si utilizásemos el dataset CelebA como pordemos ver en la figura 2.21.

DiscoGAN

La arquitectura de la red DiscoGAN tiene de nuevo el objetivo de, como la CycleGAN, descubrir relaciones entre 2 dominios a través de dos conjuntos de datos no emparejados para ser capaz de transfomar una imagen de un dominio de entrada a otra del de salida manteniendo la identidad de la imagen original [13].

De cara a conseguir esta traducción entre dominios se basa también en la reconstrucción de la imagen original para minimizar la pérdida de identidad, pero aún siguiendo el mismo principio que la CycleGAN utiliza otra arquitectura de red diferente que combina dos redes para conseguir este objetivo.

ExprGAN

Es una arquitectura de red que siguiendo el mismo concepto que la CGAN, introduce nuevos elementos para, siendo entrenada con imágenes de rostros y las respectiva etiqueta indicando la emoción que presenta, conseguir transformar la emoción de una cara a otra y con un nivel de expresividad determinado [14].

¿Por qué CycleGAN?

Después de investigar sobre las diferentes alternativas a la red CycleGAN, que es con la que íbamos a trabajar en un principio, se decidió continuar trabajando con ella por diferentes motivos. En primer lugar, se descartaron todas aquellas redes que no partían de una imagen original sino que directamente las generaban, puesto que la idea original del proyecto era esa. Después se pasaron a valorar otros aspectos como las implementaciones disponibles o los resultados que las redes eran capaces de conseguir. La IcGAN, pese a haber permitido entrenar un solo modelo para todas las emociones, carecía de un elemento como el *cycle consistency loss* que forzara la imagen final a mantener lo máximo posible la identidad de la imagen original, produciendo resultados como los vistos en la figura 2.21 que en muchos casos hacen que la persona no sea del todo reconocible. Además, habría forzado a entrenar 3 modelos por separado, los dos encoders y la propia red. Estos hechos junto a que la única implementación disponible estuviera fuertemente acoplada al dataset CelebA, terminaron descartando su uso. Por lo que respecta a la

ExprGAN, además de compartir los mismos problemas de la anterior arquitectura, su única implementación estaba aún más ligada al conjunto de datos y estaba desarrollada sobre versiones antiguas de Python y Tensorflow que dificultaban su puesta en marcha. También había multitud de *issues* abiertos en su repositorio que reportaban problemas y un incorrecto funcionamiento de la red que no estaban contestados por lo que se descartó su uso. Por último, en lo referente a DiscoGAN, pese a ser una candidata válida y con la que se podrían entrenar modelos, se decidió centrar la experimentación y el trabajo entorno a la red CycleGAN puesto que se encontraron más implementaciones disponibles que utilizaban tecnologías más familiares y que permitían de forma más sencilla entrenar con nuevos conjuntos de datos.

3. Metodología

Este capítulo tiene como objetivo reflejar el proceso de desarrollo del proyecto y para ello explicaremos en una primera sección los pasos que se han seguido y después vamos a dedicar una segunda sección a las herramientas que hemos utilizado.

3.1. Aproximación

De cara a conseguir nuestro objetivo de transformar la emoción de un rostro neutral a otra emoción de entre las 6 emociones básicas vamos a entrenar la red CycleGAN con rostros neutros y de una emoción determinada, generando un modelo para cada una de dichas emociones. Siguiendo la nomenclatura vista en la presentación teórica de la red en la sección 2.2.1, el primer dominio base X sería el de rostros con emoción neutra, y el segundo dominio objetivo Y sería el de rostros con una emoción determinada. En la figura 3.1 se ve reflejado un esquema del entrenamiento para la emoción de felicidad[1].

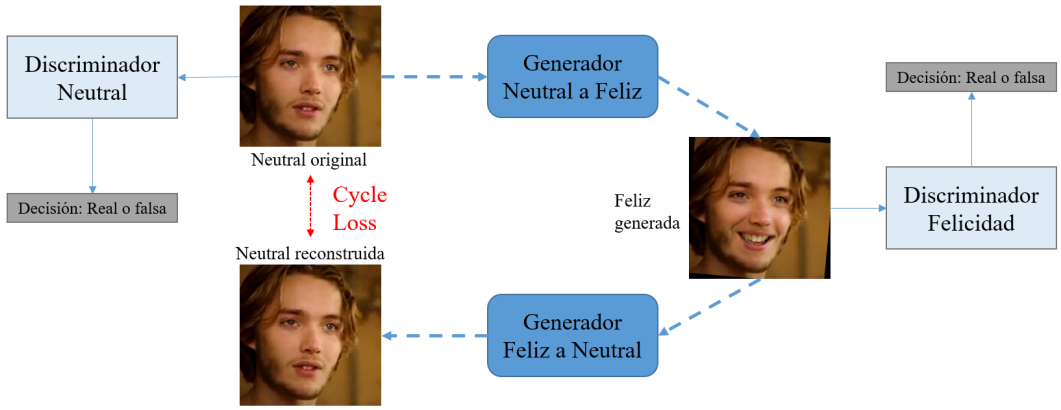


Figura 3.1.: Esquema de utilización de la CycleGAN para transformar emociones

[1] towardsdatascience.com/turning-fortnite-into-pubg-with-deep-learning-cyclegan-2f9d339dcdb0

La secuencia de pasos que se ha llevado a cabo para desarrollar el proyecto es la siguiente:

1. **Estudio de las redes GAN y de las implementaciones adecuadas para nuestro objetivo**
   En este punto se ha realizado una investigación sobre las redes GAN y los tipos de redes relacionados para conseguir entender su funcionamiento. Además, también ha llevado a cabo un estudio de las alternativas existentes para finalmente seleccionar la que mejor se acopla a nuestro proyecto.

2. **Búsqueda y análisis de los datasets existentes de emociones**
   De los puntos más importantes en un trabajo relacionado con redes neuronales son los datos que se utilizan durante el entrenamiento. Por ello se ha dedicado tiempo de desarrollo a la búsqueda de los datasets disponibles sobre emociones, a su análisis para ver los puntos fuertes de cada uno y finalmente a su selección basada en diferentes criterios para encontrar aquel que pudiera proporcionar los mejores resultados posibles.

3. **Implementación de un método de procesado para el dataset**
   También se ha estudiado estudiado e implementado un sistema de procesado de imágenes para intentar normalizar aquellas con las que se va a entrenar la red para intentar mejorar los resultados lo máximo posible.

4. **Selección de implementación de la red CycleGAN**
   De cara a comenzar el entrenamiento de nuestros modelos, se han buscado diferentes alternativas de implementaciones de la CycleGAN y se ha seleccionado una que balancease su accesibilidad con los mejores resultados. También se han instalado las dependencias que necesitaba.

5. **Entrenamiento de la red para diferentes emociones y experimentación**
   Finalmente se ha llevado a cabo un proceso de experimentación entrenando modelos para diferentes emociones y con diferentes configuraciones de entrenamiento, comparando los resultados obtenidos en unas condiciones y en otras.

3.2. Herramientas utilizadas

A continuación se van a explicar las herramientas que se han utilizado para el desarrollo del proyecto, comenzando por una descripción de las tecnologías utilizadas e incluyendo a continuación un subapartado en el que se describen las especificaciones de los ordenadores que se han utilizado.

Python

Python [2] es un lenguaje de programación de alto nivel de propósito general e interpretado que fue creado por Guido van Rossum y cuya primera versión fue lanzada en 1991. Su filosofía se basa principalmente en la legibilidad del código y soporta múltiples paradigmas de programación.

OpenCV

OpenCV o *Open source Computer Vision*[3] es una librería de programación principalmente enfocada a la visión artificial en tiempo real. Fue originalmente desarrollada por Intel. Se trata de una librería multiplataforma que por estar publicada bajo licencia BSD permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones expuestas en la misma licencia.

Está escrito en C++ y su principal API es para C++ aunque también existen interfaces para Python, Java y MATLAB que se pueden encontrar en su documentación online.

Dlib

Dlib[4] es una librería de software multiplataforma y de propósito general escrita en C++. Su diseño está furtemente influenciado por ideas del diseño por contrato o el

[2] https://www.python.org/
[3] https://opencv.org/
[4] http://dlib.net/

software basado en componentes. Se trata de un software *open-source* distribuido bajo una licencia de software Boost.

Desde que comenzó su desarrollo en 2002 se han incluido multitud de componentes en la librería que se engloban en una gran variedad de dominios desde el *networking* hasta el *Machine Learning*, pasando por otros como el algebra lineal o el procesado de imágenes.

Tensorflow

TensorFlow[5] es una biblioteca de Python para computación numérica rápida. Es una biblioteca básica que se puede usar para crear modelos de aprendizaje profundo directamente o mediante el uso de bibliotecas contenedoras que simplifican el proceso construido sobre TensorFlow.

Fue creado y mantenido por Google y publicado bajo licencia de código abierto Apache 2.0. La API fue desarrollada para lenguaje de programación Python, aunque hay acceso a la API C ++ subyacente. A diferencia de otras bibliotecas numéricas diseñadas para deep learning como Theano, TensorFlow se diseñó para su uso tanto en investigación y desarrollo como en sistemas de producción. Se puede ejecutar en sistemas de CPU individuales, GPUs, dispositivos móviles y sistemas distribuidos a gran escala de cientos de máquinas.

Git

Git[6] es un software de control de versiones que tiene como finalidad realizar un seguimiento de los cambios en el código fuente en el desarrollo de un software. Fue originalmente creado para ayudar a la coordinación entre los programadores en un proyecto pero también se utiliza para mantener un control sobre todos los cambios que se realizan a lo largo del tiempo.

Fué creado por Linus Torvalds en 2005 y se trata de un software libre distribuido bajo los términos de la licencia GNU versión 2.

[5] https://www.tensorflow.org/
[6] https://git-scm.com/

Ordenadores utilizados en el desarrollo

Durante el proceso de desarrollo se han utilizado diferentes máquinas. El portátil se ha dedicado utilizado sobretodo de cara a la implementación de los scripts y los servidores para el entrenamiento de modelos.

- **Servidor 1 - *Jackson***
  - **SO**: Ubuntu 16.04
  - **CPU**: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
  - **RAM**: 16GB DDR4
  - **GPU 1**: NVIDIA QUADRO P6600
  - **GPU 2**: NVIDIA RTX 2080ti
- **Servidor 2 - *Oprah***
  - **SO**: Ubuntu 16.04
  - **CPU**: Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz
  - **RAM**: 16GB DDR4
  - **GPU 1**: NVIDIA GTX 1080ti 12GB
  - **GPU 2**: NVIDIA TITAN Xp 12GB
- **Portátil**
  - **SO**: Ubuntu 18.04
  - **CPU**: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
  - **RAM**: 8GB DDR4
  - **GPU**: NVIDIA GTX 1050

4. Desarrollo

En este capítulo se va a explicar como se han llevado a cabo los diferentes pasos necesarios para completar el proyecto y el razonamiento detrás de algunas decisiones tomadas durante el mismo.

4.1. Dataset

De cara a plantear un trabajo en el campo de las redes neuronales, el punto principal a valorar y del que en gran parte van a depender los resultados es el del conjunto de datos del que disponemos para el problema sobre el que intentamos plantear una solución. En el caso del trabajo que nos ocupa, se han buscado datasets existentes que contengan imágenes de caras de personas etiquetadas con la emoción que reflejan en su rostro.

4.1.1. Conjuntos encontrados

En esta subsección se van a comentar diferentes aspectos sobre los conjuntos imágenes que se han explorado y con los que se ha trabajado, explicando por qué motivos se ha decidido utilizar unos u otros. Es importante destacar que aún tratándose de datasets orientados a una sola problemática, los conjuntos presentan imágenes de carácter en ocasiones muy diferente [15] y distribuyen los datos de manera heterogénea.

- **Oulu-CASIA dataset**
  Este dataset fue solicitado a Dr. Guoying Zhao [16] por ser utilizado en una red GAN que tenía un propósito similar a la nuestra, la ExprGAN, como se ha comentado en la seccion 2.2.2. Presentaba características interesantes como una serie de muestras para diferentes valores de expresividad para cada sujeto y expresión. Sin

embargo, por el reducido número de sujetos y la falta de muestras de ciertas expresiones para muchos de ellos, así como los marcados rasgos asiáticos en la mayoría de sujetos y la falta de calidad en la mayoría de las imágenes se descartó su uso.

(a) (b) (c) (d) (e)

Figura 4.1.: Muestras de Oulu-CASIA dataset

- **Cohn-Kanade AU-Coded Facial Expression Database[17]**
  De igual manera que en el caso anterior este dataset fue confeccionado en un entorno controlado y está compuesto por imágenes para cada emoción posadas por 123 sujetos. En este caso los modelos están distribuidos de manera más uniformemente en el espacio de edades y para ambos sexos. Además de emociones posadas, también están incluidas un conjunto de imágenes obtenidas mediante la exposición de los sujetos a diversos contenidos que tenían el objetivo de generar determinadas emociones y, tanto para este último conjunto como para el anterior, se registraron también las imágenes con un ángulo de 30 grados.

  Para cada imagen se proporciona su correspondiente etiqueta con la emoción registrada así como las AUs o *action units* activas. Este último tipo de notación se explicará más adelante.

(a) (b) (c)

Figura 4.2.: Muestras de Cohn-Kanade AU-Coded Facial Expression Database

- **FER2013**
  Este dataset es reconocido por ser uno de los más usados en el ámbito del reconocimiento de expresiones faciales. Se trata de un extenso y público conjunto de imágenes que por primera vez a diferencia de los casos anteriores presenta rostros

que se corresponden con personas que varían de manera significativa en edad y postura entre otros. Está compuesto por 35887 recortes de caras con un procesado con el objetivo de realizar ese recorte y convertir la imagen a escala de grises.

También como diferencia a los datasets comentados con anterioridad, distribuye los datos a través de un fichero CSV en el que la primera columna representa la expresión, la segunda tiene los valores de todos los píxeles de la imagen y la tercera indica que uso dar a dicha imagen, entrenamiento, validación o test. De esta manera es el único conjunto de datos con el que se ha trabajado que no distribuye archivos de imagen propiamente dichos.

(a) (b) (c) (d)

Figura 4.3.: Muestras de FER2013 dataset

- **EmotioNet**
  Se trata de un conjunto de datos confeccionado para la competición de su mismo nombre [18], consistente en dos partes: por una parte el reconocimiento de expresiones faciales y por otra el reconocimiento de las AUs o *action units* activas en un rostro. Está conformado por un millón de imágenes que, de la misma forma que en el caso anterior, prescinden de las condiciones ideales de los conjuntos de imágenes preparados en laboratorios con la ayuda de actores y proponen una búsqueda del reconocimiento de expresiones *in the wild*.

AU #	Acción	AU #	Acción
1	inner brow raiser	2	outer brow raiser
4	brow lowerer	5	upper lid raiser
6	cheek raiser	9	nose wrinkler
12	lip corner puller	17	chin raiser
20	lip stretcher	25	lips part
26	jaw drop	-	-

Tabla 4.1.: AUs utilizados en EmotioNet

El conjunto de imágenes en este caso es también distribuido a través de una hoja de datos pero en este caso la propia imagen se proporciona a través de una URL.

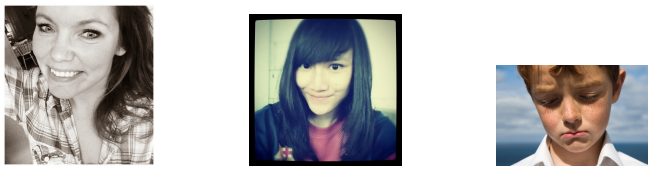
Emocion	AUs
Felicidad	12, 25
Tristeza	4, 15
Miedo	1, 4, 20, 25
Rabia	4, 7, 24
Sorpresa	1, 2, 25, 26
Asco	9, 10, 17

Tabla 4.2.: Asociación entre AUs y emociones

Además, en lugar de otorgar un único valor a la imagen para describir la emoción que presenta, el conjunto de datos utiliza la relación existente entre las dos tablas anteriores para describirla. De esta manera, para un rostro de felicidad representativo los valores 12 y 25 tendrían un valor de uno mientras que los demás estarían a cero.

Esta notación permite una descripción más exacta y objetiva de las imágenes pero complica la tarea a la hora de clasificar las imágenes puesto que en el caso de que la emoción no se vea claramente representada puede darse el caso de que existan solapamientos entre las *action units* y para una imagen el propio etiquetado no tenga claro qué emoción se está viendo representada.

Además, del total de imágenes solo 50000 han sido anotadas manualmente por expertos y 950000 lo han sido por un algoritmo con una precisión del 81 %.



(a) (b) (c)

Figura 4.4.: Muestras de EmotioNet dataset

- **AffectNet[19]**
  El último dataset que se va a comentar en esta sección sigue la corriente de los últimos dos al volver a ser considerado como un conjunto de imágenes que representan emociones *in the wild*. Sin embargo en este caso, junto con dicho conjunto de cerca de un millón de imágenes de rostros heterogéneos, se proporcionan también en este caso la posición de la misma cara así como sus dimensiones obtenidas

por el detector de OpenCV y también la de sus 68 *landmarks* faciales.

Del total de un millón de imágenes, 450000 han sido etiquetadas manualmente por un conjunto de doce expertos tanto de manera categórica, etiquetando cada rostro con un valor asociado a una de las 6 expresiones básicas; como de manera dimensional, proporcionando 2 valores: *Valence*, que representa cómo de positivo o negativo es un evento causante de la emoción, y *Arousal* que representa la expresividad mostrada.

(a) (b) (c)

Figura 4.5.: Muestras de AffectNet dataset

4.1.2. Análisis de los datasets encontrados

Una vez se han encontrado y explorado los conjuntos de datos aplicables al trabajo que se esta llevando a cabo, es necesario entrar a valorarlos y compararlos teniendo en cuenta la aplicación objetivo de tal forma que podamos encontrar aquel o aquellos que presentan las características que a priori deberían proporcionarnos mejor resultado, permitiéndonos así empezar los primeros experimentos.

Con este fin pues podemos sintetizar los requisitos que buscamos de la siguiente manera:

- Basados en los datasets con los que se han utilizado la red CycleGAN:
  - Número de imágenes de cada clase elevado
  - Balance entre imágenes de las dos clases
  - Imágenes cuadradas
- Basados en preferencias del proyecto:
  - Imágenes a color
  - Diversidad en el género, edad y origen de los rostros
  - Fiabilidad del etiquetado
  - Comodidad de uso

Determinados los criterios anteriores podemos empezar a analizar propiamente los conjuntos del apartado previo.

Del conjunto de requisitos, aquel que se demuestra más restrictivo es el de que el número imágenes para cada clase sea elevado puesto que si lo aplicamos y ponemos el umbral entorno a las 1000 imágenes basándonos en el dataset que los autores de la red utilizan para probar la red (horse2zebra) eliminaríamos las dos primeras opciones correspondiente con aquellos conjuntos que han sido obtenidos en laboratorio, sugiriendo que los conjuntos que registran emociones *in the wild* pueden dar mejor resultado.

Los tres conjuntos restantes presentan características similares. Sin embargo, cabe destacar que el FER2013 proporciona las imágenes en escala de grises por lo que no se ajusta a nuestras preferencias. Este hecho junto a que la resolución con la que las distribuye propicia fallos en el detector de *landmarks* faciales que se utilizará en el preprocesado de las imágenes, alejan a este dataset de ser seleccionado como el más favorable para nuestro problema.

Los dos últimos datasets que quedan pues serían EmotioNet y AffectNet, cumpliendo ambos con todos criterios clave: disponen de un gran número de imágenes, podemos disponer de dos conjuntos balanceados de emociones ya que de aquella emoción de la que menos imágenes se dispone en ambos casos cumple con el mínimo umbral del que hemos hablado anteriormente, y ambos también nos permiten obtener imágenes cuadradas de rostros. Por lo que respecta a los criterios propios del proyecto, ambos conjuntos también los cumplirían en su integridad. Por este motivo, de cara a seleccionar cuál de ellos iba a tener mayor preferencia en la experimentación se recurrió a la evaluación cuantitativa de algunos de estos requisitos puesto que, pese a que ambos los cumplen, es posible decir que en nuestro caso uno, AffectNet, lo hace de una manera mejor para nuestro trabajo.

El primero de estos requisitos es el de la fiabilidad en el etiquetado, y es que pese a que ambos datasets contengan un número de imágenes similar, AffectNet proporciona cerca de 450000 etiquetadas manualmente por expertos mientras que EmotioNet solamente 50000, delegando el resto a un algoritmo con una precisión del 81 % como se ha comentado en la sección anterior. Este hecho decanta la elección hacia el primer conjunto puesto que presenta mayor fiabilidad para una misma cantidad de imágenes.

El segundo de los criterios por otra parte es el de la comodidad de uso. En este caso AffectNet se sobrepone a EmotioNet de dos maneras: por una parte, el etiquetado de las expresiones se corresponde de manera directa con el que necesitamos por lo que facilitará la tarea de gestionarlo, de igual forma que nos librará de los problemas que la notación del segundo dataset presenta y que se han comentado en su apartado; por otra parte, al distribuir las imágenes de manera directa y no por medio de URLs nos facilita la tarea de recopilarlo.

De esta manera se puede concluir que el conjunto de datos más favorable para nuestro problema y en el que deberemos centrar nuestra experimentación es AffectNet, ya que tal y como hemos podido comprobar es el que mejor se adapta a nuestras exigencias y sobre todo a las del problema.

4.1.3. Balanceado de dataset

Una vez se completó el trabajo relacionado con la selección del dataset detallado en la sección anterior, se tiene que evaluar las condiciones del conjunto de datos elegido y comenzar con su procesado con el objetivo de adaptarlo a las exigencias del problema. De esta manera se analizó la distribución del total de las imágenes entre las diferentes emociones.

Emoción	Número de imágenes
Neutral	75,374
Felicidad	134,915
Tristeza	25,959
Asco	4,303
Sorpresa	14,303
Furia	25,382

Tabla 4.3.: Distribución de imágenes en AffectNet

La distribución irregular de las imágenes viene dada por la naturaleza de las mismas: Las imágenes de este dataset han sido obtenidas de internet y etiquetadas a mano por lo que no ha sido controlada su captura para tratar de tener una distribución equilibrada, a diferencia de otros casos en los que las fotografías han sido tomadas por el mismo equipo que confecciona el dataset y se ha hecho posar a los sujetos en todas las posibles poses para que el conjunto de esta manera sí quede finalmente uniformemente distribuido.

Puesto que de cara a entrenar la red con la que se iba a trabajar necesitábamos conjuntos de imágenes de tamaño similar para cada expresión se decidió utilizar los datos proporcionados por el equipo que confeccionó el dataset para realizar un filtrado con el objetivo de quedarnos con las mejores imágenes posibles para cada emoción. Estos datos a parte de un valor que representa la emoción presente la imagen incluían también los valores de *Valence* y *Arousal* que representan la valencia y la expresividad presentes en la misma y este último ha sido el empleado para llevar a cabo este balanceado.

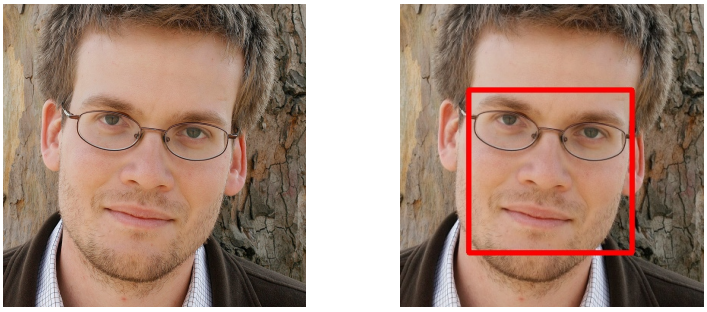
Utilizando este dato se implementó un *script* que, para cada expresión, permite alterar el umbral para este valor y controlar la cantidad de imágenes que se obtenían, asegurándonos de esta manera de que el conjunto de imágenes obtenido sería el más representativo posible puesto que nos interesaba que las muestras finales con las que íbamos a entrenar intentasen maximizar este valor. Dependiendo de los requisitos específicos del entrenamiento de la experimentación que estuviésemos llevando a cabo se utilizó este

programa para obtener los conjuntos de imágenes del tamaño que se precisaba.

4.1.4. Procesado de dataset

La imágenes presentes en el AffectNet debido a que han sido extraídas de internet y no de un entorno supervisado como se ha comentado en la sección anterior, no siempre incluyen los rostros que son la parte que más nos interesa de la imagen en la misma posición, en la misma orientación o con el mismo tamaño. Puesto que de cara a entrenar la red interesa proporcionar los datos de la manera más uniforme posible, se ha propuesto un preprocesamiento para las imágenes a utilizar que se va a detallar a continuación.

El primer paso de este proceso trata de solucionar el problema de conocer en qué lugar se encuentra el rostro que se busca. Para ello, se ha recurrido al detector de rostros proporcionado por la librería dlib. De esta manera para cada imagen se es capaz de obtener la posición de los rectángulos que actúan de *bounding box* para todas las caras presentes en la imagen, que en el caso de usarlo sobre las imágenes de AffectNet siempre es una mientras no se den falsos positivos.



(a) Imagen original (b) Bounding box

Figura 4.6.: Detección de cara

El siguiente problema a resolver una vez se conoce la localización del rostro es el de la orientación. En este caso se ha utilizado el detector de 5 puntos característicos de caras también incluido en dlib sobre la zona determinada en el punto anterior para tratar de alinear la cara presente en ella. Estos cinco puntos característicos se han utilizado para calcular los puntos medios de los ojos y, a través de estos, conocer el ángulo de inclinación presente. Con dicho ángulo y a través de la función de *rotate* de la librería

imutils[1] se completa la rotación de la imagen.

(a) Imagen original (b) Landmarks detectados (c) Imagen rotada

Figura 4.7.: Proceso de rotación

Por último, para obtener la imagen final con la que se entrenará la red se utiliza la posición determinada en el primer paso para, sobre la imagen ya rotada, recortarla haciendo que en dicha imagen final solo quede el rostro. Puesto que el rectángulo calculado por el detector de dlib en algunos casos es demasiado ajustado, se ha introducido un margen cuyo valor ha sido obtenido por experimentación para tratar de no eliminar información de la cara, aunque siempre tratando de aislar el rostro lo máximo posible.

(a) Imagen rotada (b) Imagen final

Figura 4.8.: Recorte final

[1] https://pypi.org/project/imutils/

4.2. Selección de implementación de CycleGAN

De cara a comenzar el proceso de experimentación y comprobar qué resultados somos capaces de conseguir entrenando nuestra red con el dataset que hemos elegido, falta todavía escoger qué implementación de la red CycleGAN vamos a utilizar. Este paso resulta de vital importancia puesto que nuestros resultados van a depender en gran medida de la que utilizemos.

4.2.1. Criterios de selección

Los criterios que se han valorado en este proceso de selección han sido los siguientes:

- Rendimiento
- Tecnologías utilizadas
- Accesibilidad

En cuanto al rendimiento, se valorará que la implementación consiga un balance entre la calidad de los resultados conseguidos y el tiempo que utiliza para conseguirlos. Sin embargo, tendrá prioridad en este aspecto la calidad final y solo será en el caso de que se consigan resultados de calidad similar cuando se valorará el tiempo requerido.

En lo referente al segundo punto, se va a buscar que la implementación se haya llevado a cabo bajo tecnologías con las que se haya trabajado previamente, con el objetivo de poder tener más control sobre la misma y modificarla en el caso que sea necesario. Además, esta familiaridad también repercutirá de manera positiva en la configuración del entorno en el que se va a utilizar la red. En este caso por ejemplo, se preferirá una red desarrollada con Keras que directamente utilizando Tensorflow, y se preferirá que esté desarrollada utilizando Tensorflow antes que mediante PyTorch.

El último apartado que se va a tener en cuenta es este proceso de selección es el de accesibilidad y en este caso se pretende buscar una implementación que ofrezca simplicidad a la hora de ser utilizada y acceso directo a los parámetros de configuración del entrenamiento.

4.2.2. Implementaciones encontradas

Se han encontrado multitud de repositorios que contienen implementaciones de la CycleGAN, y se han utilizado los comentarios sobre las mismas para realizar un filtrado de aquellas que presentan problemas. Después de este filtrado, se han utilizado los criterios del punto anterior para valorar qué red nos convenía utilizar y tras contrastar diversas redes finalmente se han reducido las alternativas a 2 candidatas principales: Una primera implementación en Keras[20] que cumpliría los puntos de accesibilidad y tecnologías utilizadas, y otra en Tensorflow[21] que pese a reducir la accesibilidad y tratarse de una tecnología con la que se dispone de menos experiencia, promete mejores resultados.

Para tomar la decisión final se han descargado ambas implementaciones y se ha realizado un entrenamiento con los parámetros por defecto para evaluar los resultados que consiguen cada una. Vamos a utilizar uno de los datasets que utilizan los creadores de la red para demostrar sus resultados, horse2zebra, asegurándonos de que las redes deberían de conseguir buenos resultados.

Antes de realizar ambos entrenamientos, se ha preparado el servidor Oprah del laboratorio instalando las dependencias de cada red en dos entornos virtuales diferentes. Dichas dependencias están especificadas en los correspondientes repositorios de cada red.

Resultados de entrenamiento

A continuación vamos a presentar los resultados que se han obtenido con esos entrenamientos a través de una serie de muestras tomadas en diferentes momentos del entrenamiento y el tiempo que ha requerido cada uno; y se va a justificar la selección final de una implementación.

Implementación	Tiempo utilizado
Keras	7:36:08
Tensorflow	18:45:24

Tabla 4.4.: Tiempo utilizado por cada implementación para el entrenamiento en la GTX 1080ti del servidor Oprah (3.2)

Los resultados de los entrenamientos llevados a cabo reflejados en las figuras 4.9 y 4.10 reflejan cómo la implementación que utiliza Tensorflow es capaz de producir las transformaciones que buscamos y conseguir buenos resultados. La red implementada en Keras por el contrario es incapaz de realizar dicha transformación. Por este motivo, y pese a que el tiempo que ha utilizado la implementación en Tensorflow es mucho mayor, se ha elegido esta para comenzar a experimentar con nuestros conjuntos de datos.

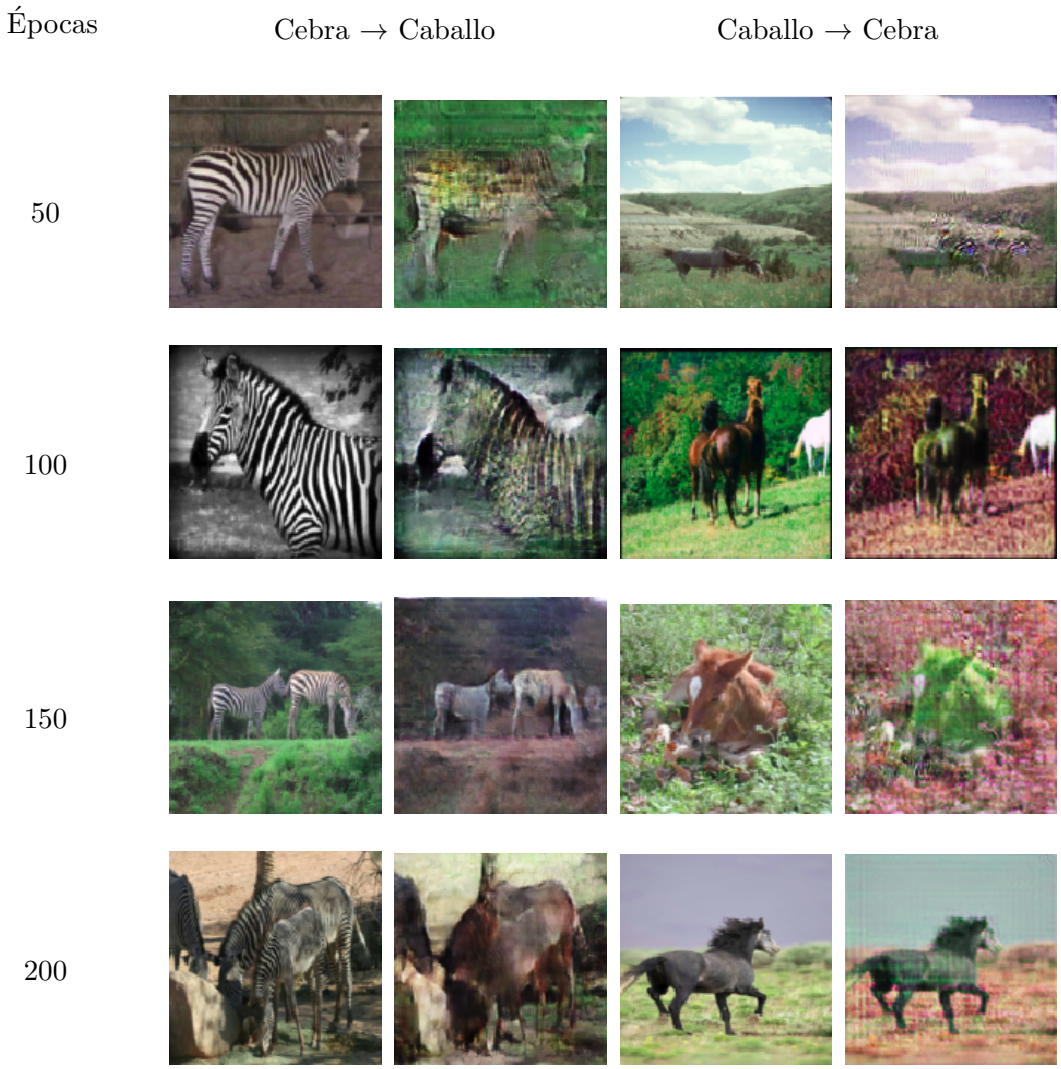


Figura 4.9.: Muestras del entrenamiento de la implementación en Keras

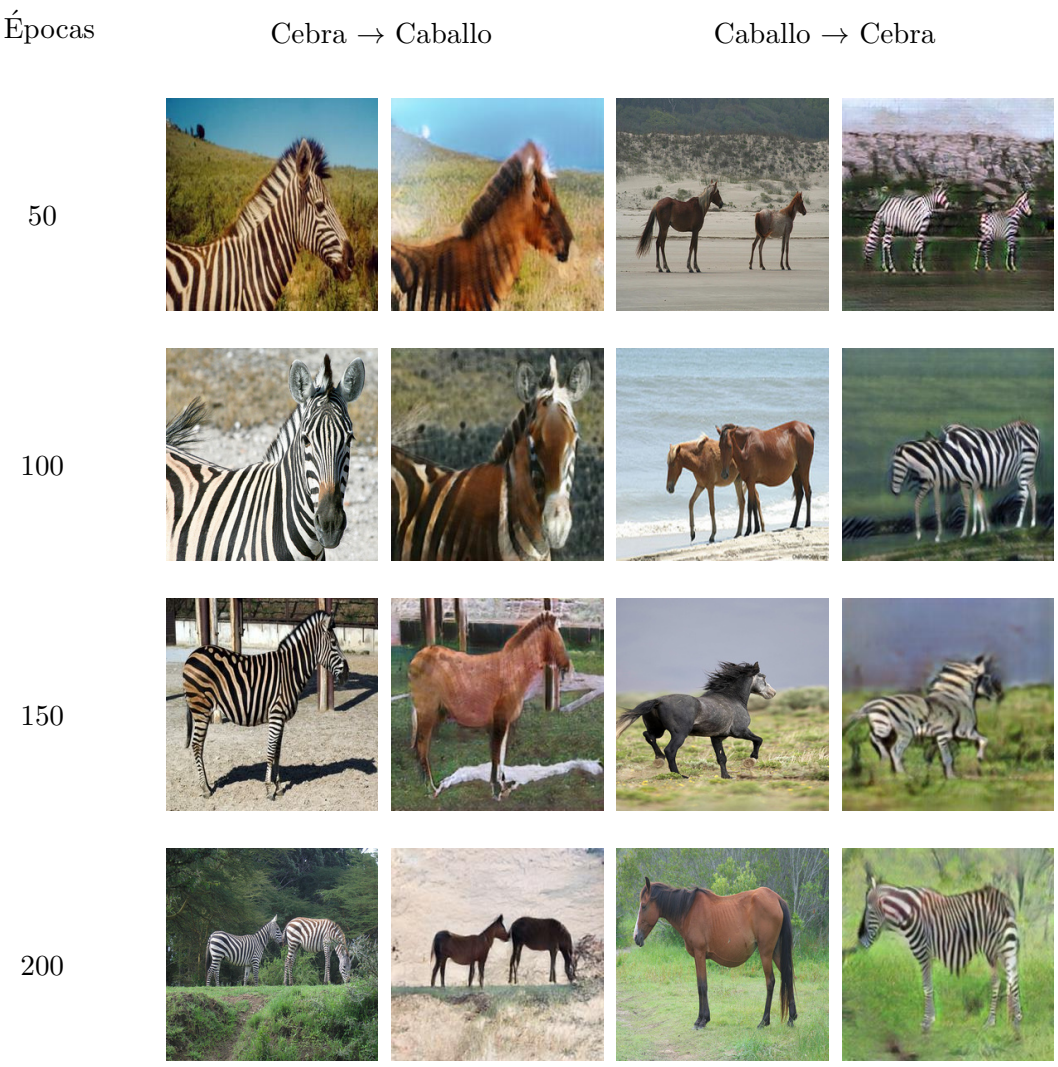


Figura 4.10.: Muestras del entrenamiento de la implementación en Tensorflow

5. Experimentación

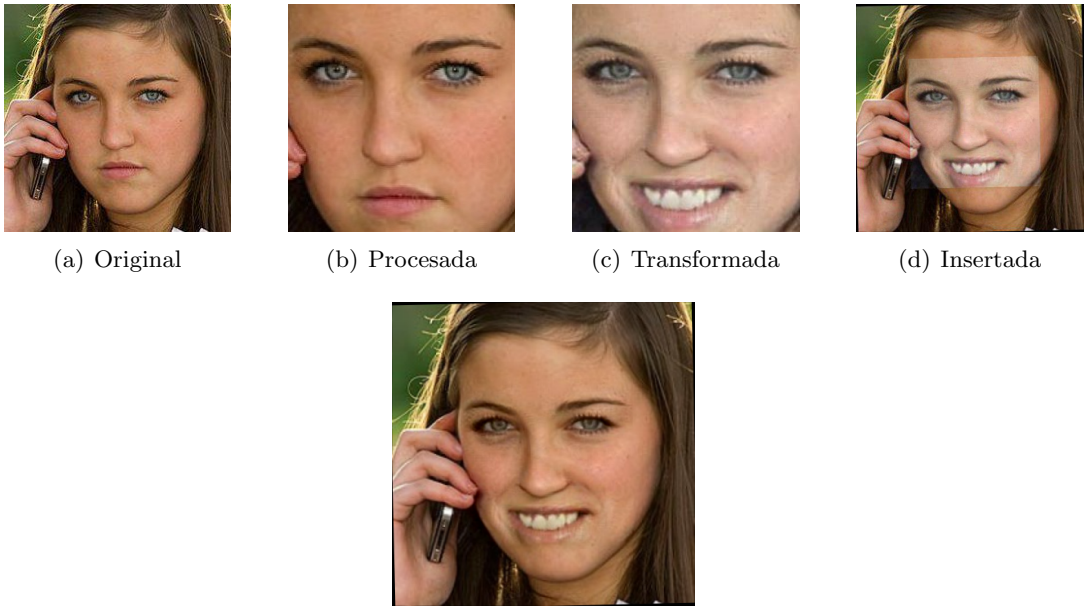
En este capítulo se va a realizar un recorrido por el proceso de experimentación que se ha llevado a cabo y se van a presentar los resultados conseguidos a través de diferentes entrenamientos que se han realizado.

5.1. Primer entrenamiento

Una vez hemos encontrado el dataset que mejor se adapta a nuestro problema, hemos propuesto e implementado un método de procesado para el mismo y hemos decidido qué implementación de la red vamos a utilizar, estamos en disposición de comenzar a entrenar modelos.

La primera emoción con la que se va a trabajar es la de felicidad por dos motivos principales. El primero es que por ser la emoción que presenta los rasgos más representativos, resulta fácil pensar que será la que menos problemas suponga a la red a la hora de identificar las características identificativas de cada dominio, como en este caso sería la sonrisa. Además, es de emoción de la que más imágenes se dispone por lo que probablemente resulte más sencillo encontrar imágenes más representativas que en los otros casos.

Los parámetros con los que se va a realizar el entrenamiento son los parámetros por defecto de la implementación por lo que el entrenamiento se prolongará 200 épocas. De cara a emular las condiciones en las que se ha demostrado que la red funciona adecuadamente, se utilizarán también para el entrenamiento aproximadamente 1000 imágenes de emoción neutra y 1000 de felicidad, igualando los números al dataset horse2zebra con el que se ha trabajado anteriormente. De cara a obtener esta cantidad de imágenes se ha utilizado el proceso detallado en la sección 3.1.3. En lo referente al conjunto de test



(a) Original (b) Procesada (c) Transformada (d) Insertada

(e) Imagen final

Figura 5.1.: Proceso de reconstrucción

sobre el que se valorará el resultado final del entrenamiento, se ha hecho una selección de imágenes que se mantendrá para el resto del proceso de experimentación y serán utilizadas para comparar los resultados de todos los experimentos.

Antes de mostrar los resultados finales de este entrenamiento es necesario aclarar que de cara a comparar las imágenes originales con el resultado de la red y teniendo en cuenta que en el procesado previo de las imágenes se realiza un recorte de la imagen original, se ha llevado a cabo un proceso de reconstrucción de la imagen final como se puede ver en la 5.1. Este proceso consiste en revertir los pasos detallados en 3.1.4. Además y puesto que la red modifica ligeramente la textura e iluminación de algunas imágenes, se ha utilizado la función *seamlessClone* de OpenCV para suavizar la imagen final y revertir ese cambio.

Como se puede comprobar en los resultados presentados en la figura 5.2, la red efectivamente es capaz de identificar los rasgos característicos de un rostro feliz e intenta trasladarlos a las caras neutras. Sin embargo, en este primer entrenamiento se ven reflejados tres problemas principales. El primer problema es el cambio de iluminación presente en las imágenes transformadas, tendiendo estas a mostrar una apariencia más pálida,

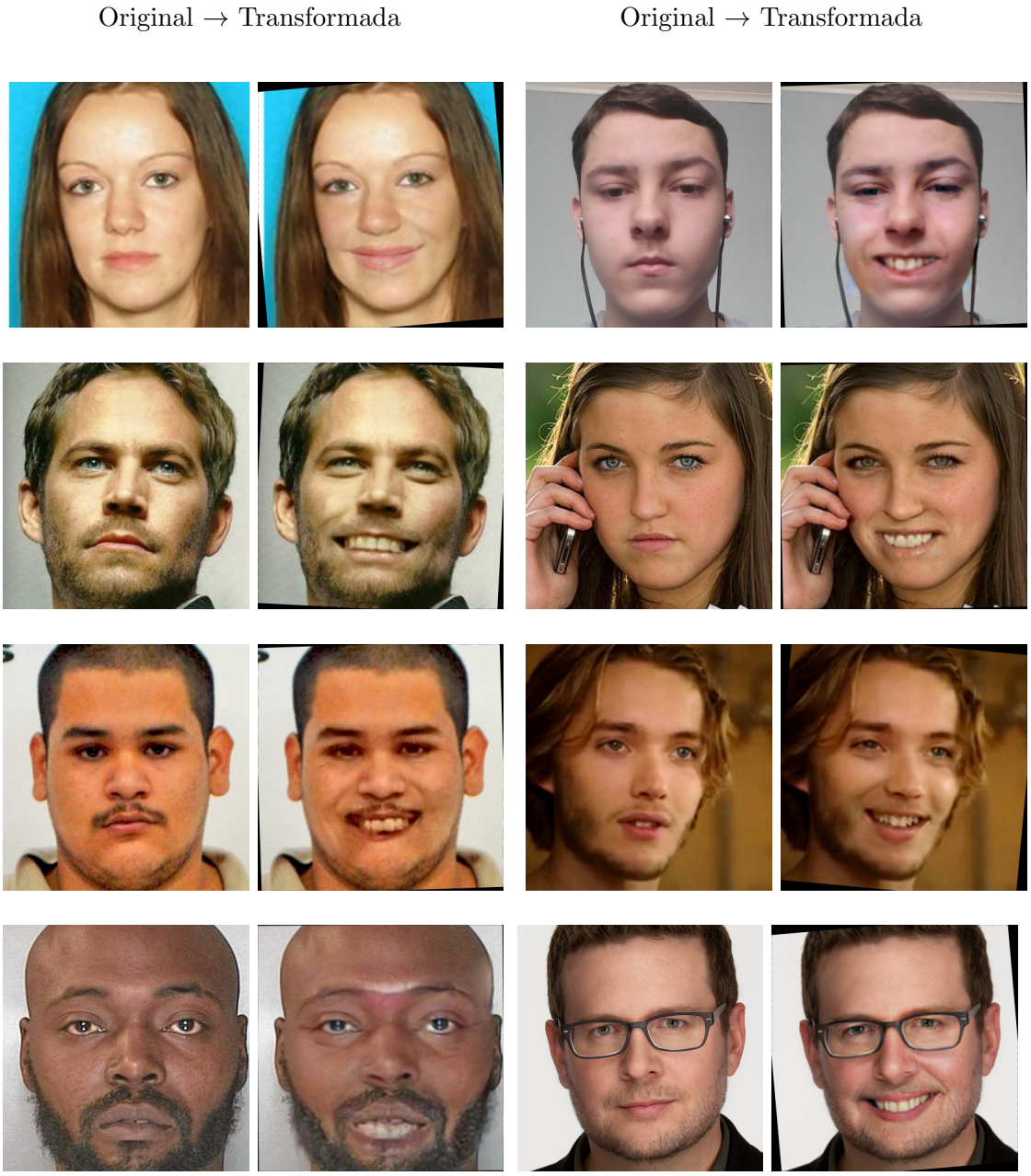


Figura 5.2.: Resultados del entrenamiento con 1000 imágenes para la emoción de felicidad y 200 épocas

aunque quede casi completamente oculto por el proceso de reconstrucción (en la figura 5.1 se puede apreciar este cambio de tonalidad entre las subfiguras b y c). El segundo problema está relacionado con la naturalidad de las expresiones. La red tiende a intentar incluir una sonrisa con los dientes visibles y en algunas situaciones o bien no es el tipo de sonrisa más adecuada para la imagen o bien no consigue añadirla de la forma más natural. El último problema está relacionado con los colectivos que no tienen muchas muestras en el conjunto de datos, haciendo que la red no sea capaz de transformar su expresión como se puede ver en el penúltimo ejemplo de la muestra.

5.2. Entrenamiento con más imágenes

Con el objetivo de intentar mejorar los primeros resultados y tratar de solucionar los problemas encontrados, se opta por incrementar el número de imágenes que se van a utilizar en el entrenamiento. El entrenamiento de esta manera se realizará de nuevo durante 200 épocas y con los mismos parámetros que en el caso anterior y en este caso se utilizarán 2000 imágenes para cada clase en lugar de 1000. Además, las nuevas imágenes para el conjunto de felicidad serán seleccionadas con un valor de expresividad menor para que la red sea capaz de generar sonrisas sin dientes de forma más natural.

Los resultados presentados en la figura 5.3 reflejan cómo esta vez el entrenamiento ha conseguido generar imágenes mucho más realistas y naturales. La mayor cantidad de imágenes ha permitido que la red entienda que la iluminación total de la imagen no es un factor representativo de los dominios entre los que estamos intentando transformar y las imágenes finales presentan una tonalidad mucho más parecida a la original. El nuevo modelo también es capaz de paliar casi de manera total el segundo problema, consiguiendo en los resultados finales unas expresiones mucho más naturales que con el anterior. Sin embargo, estas nuevas imágenes no consiguen paliar el tercer problema y la red sigue siendo incapaz de modificar de forma representativa las imágenes de colectivos menos representados en el dataset. Como solución a este problema se propone una búsqueda e incorporación de imágenes de estos colectivos de manera específica a los conjuntos de entrenamiento.

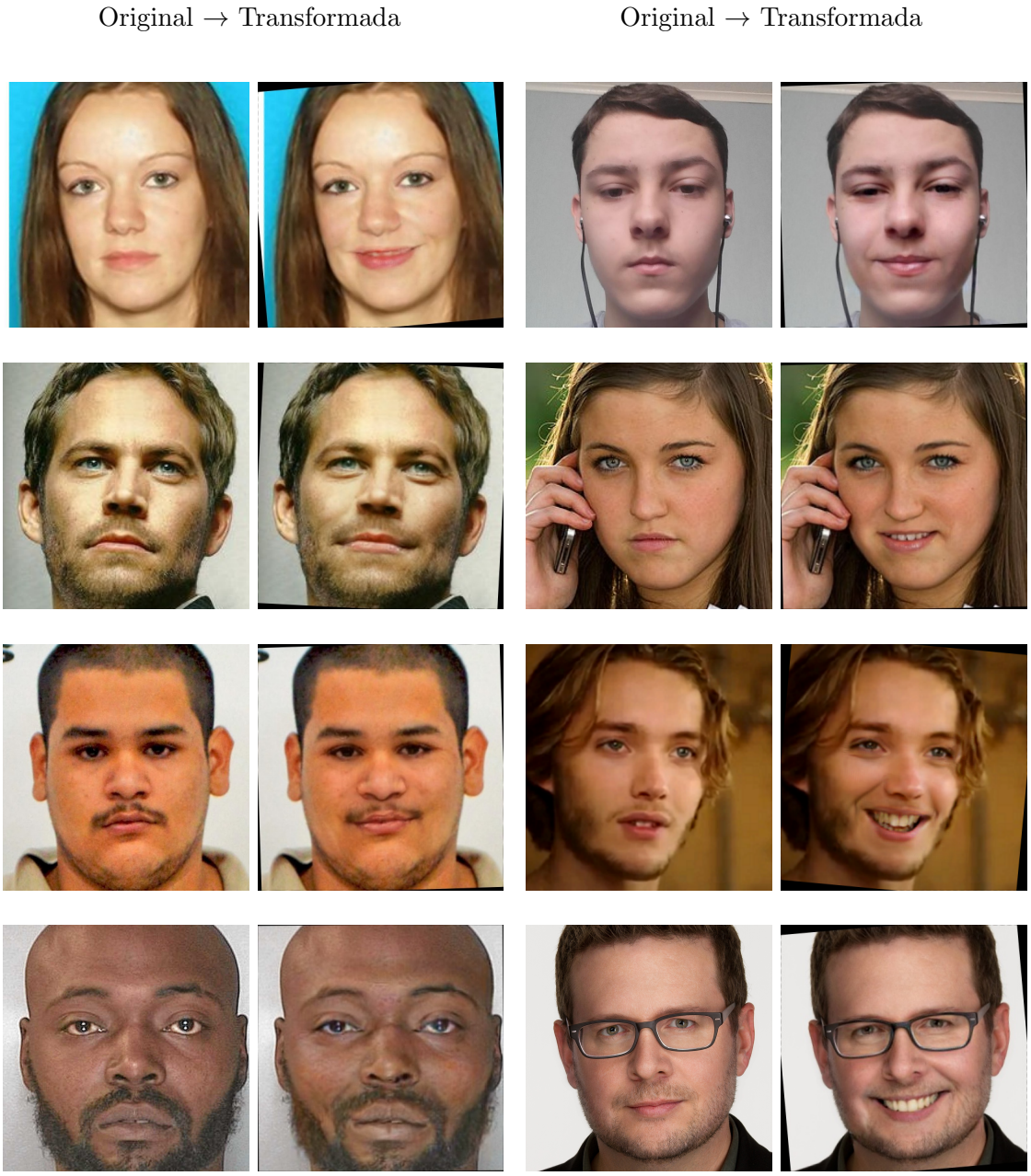


Figura 5.3.: Resultados del entrenamiento con 2000 imágenes para la emoción de felicidad y 200 épocas

5.3. Imágenes sin procesar contra imágenes procesadas

Para comprobar si el método de procesado de las imágenes que se ha detallado en la sección 4.1.4 era necesario y efectivamente conseguía mejorar los resultados se ha realizado un entrenamiento bajo las mismas condiciones que el anterior y utilizando exactamente las mismas imágenes pero sin aplicarles el procesado previo al entrenamiento.

Los resultados reflejados en la figura 5.4 muestran cómo este modelo entrenado sin un procesado previo de los conjuntos de imágenes es incapaz de acercarse a los resultados obtenidos por aquellos modelos que sí lo aplicaban. La variedad de fondos, colores de pelo, peinados, elementos secundarios y la propia posición e inclinación de la cara en la imagen afectan mucho a los resultados que la red es capaz de conseguir. La presencia de estos factores externos repercute en una dificultad mayor de cara a identificar las características de los dos dominios entre los que se está realizando la traducción e implica una distorsión de forma y textura en la imagen final que las aleja mucho de las que se han conseguido con otros modelos.

	Imágenes	Épocas	Procesado	Tiempo consumido (hh:mm:ss)
Entrenamiento 1	∼1000	200	Sí	28:03:15
Entrenamiento 2	∼2000	200	Sí	36:01:24
Entrenamiento 3	∼2000	200	No	95:56:03

Tabla 5.1.: Tiempo consumido para cada entrenamiento

Además, en la tabla 5.3 podemos comprobar que este último entrenamiento se ha prolongado durante más tiempo que los anteriores a pesar de estar utilizando el mismo número de imágenes y épocas que en el anterior. Esta demora temporal se debe a que las imágenes, al no haber sido procesadas ni reescaladas, mantienen su tamaño original. Este hecho unido a que la implementación de la red no carga las imágenes en memoria si no que las está leyendo de disco continuamente, ralentiza severamente el entrenamiento al estar leyendo y cargando continuamente imágenes de gran tamaño. De cara a reducir el tiempo de entrenamiento de los futuros entrenamientos, se propone reescalar las imágenes al tamaño que utiliza la red en el procesado propuesto y eliminar el reescalado que se realiza en la propia red.

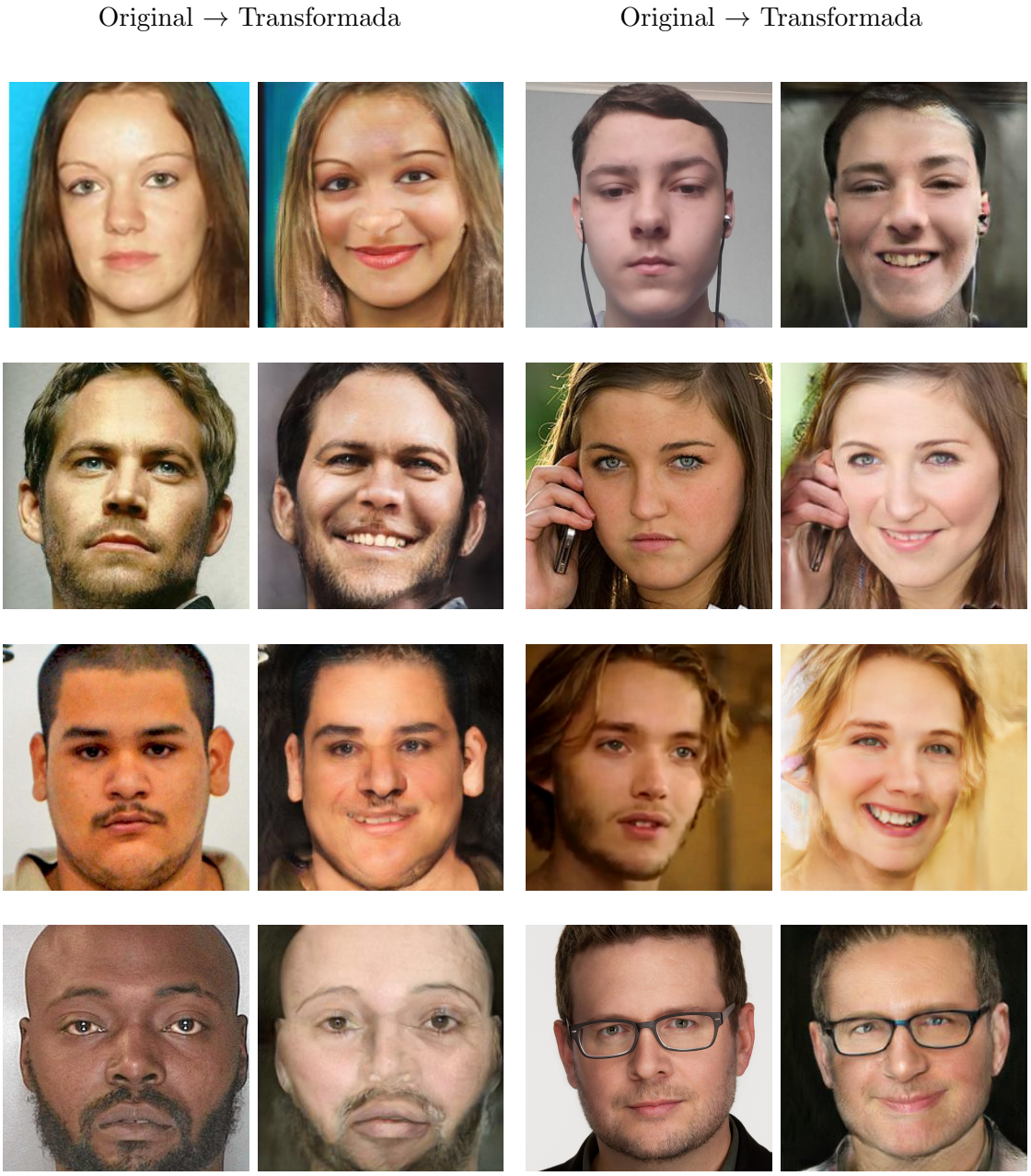


Figura 5.4.: Resultados del entrenamiento con 2000 imágenes para la emoción de felicidad sin procesar y 200 épocas

5.4. Entrenamientos para otras emociones

Una vez comprobado que la red es capaz efectivamente de reconocer las características que representan la emoción de felicidad y transformar los rostros a esa emoción manteniendo la identidad del rostro, y habiendo demostrado que el procesado previo de las imágenes ayuda enormemente a conseguir un mejor resultado final, llega el momento de entrenar modelos para conseguir inferir otras emociones y comprobar si los resultados conseguidos son tan positivos como con la primera emoción. Las emociones con las que se va a trabajar van a ser las de asco y las de sorpresa puesto que son aquellas de las que se suele disponer de menos imágenes en los datasets existentes e interesa valorar los resultados que se pueden obtener. Se realizarán entrenamientos para cada una de ellas en las mismas condiciones que el segundo entrenamiento: 200 épocas y aproximadamente 2000 imágenes por clase para el entrenamiento.

Los resultados que se pueden observar en las figuras 5.5 y 5.6 para las emociones de sorpresa y asco respectivamente muestran como la red, de igual manera que para la emoción de felicidad, efectivamente es capaz de identificar rasgos característicos de las diferentes emociones con las que estamos trabajando. Para la emoción de sorpresa el modelo entrenado reconoce la boca abierta y los ojos más abiertos de lo normal, mientras que para el asco aprende a entrecerrar los ojos, remarcar las arrugas de expresión y modificar ligeramente la forma de la boca. Sin embargo, algunas de las imágenes finales registran un transformación de dichos rasgos mucho más sutil que en el caso de la sonrisa para la emoción de felicidad, resultando en ciertos casos insuficiente para reconocer la nueva emoción en la imagen final. Además, el problema que se ha mantenido durante los entrenamientos anteriores relacionado con los colectivos minoritarios sigue presente.

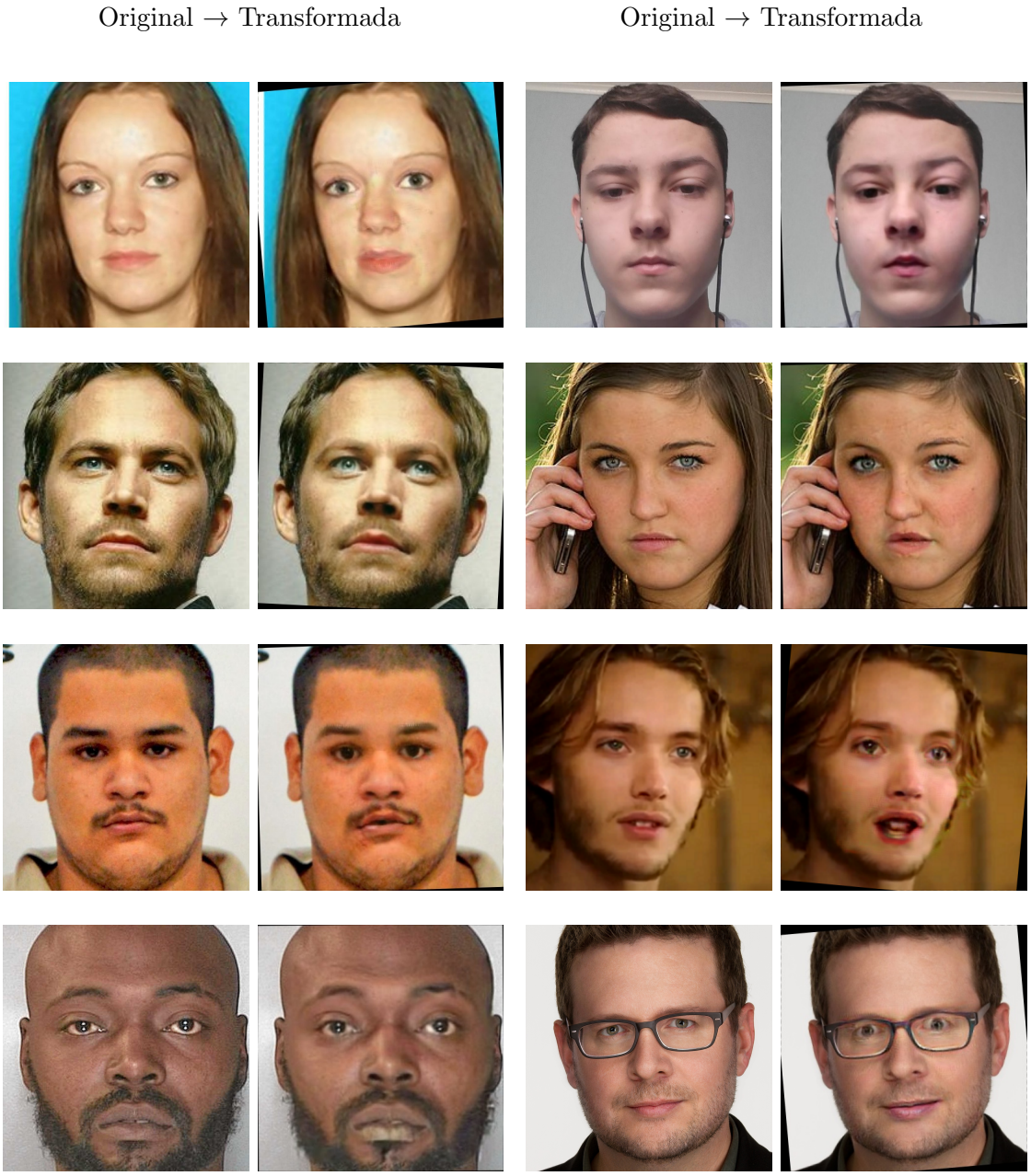


Figura 5.5.: Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 200 épocas

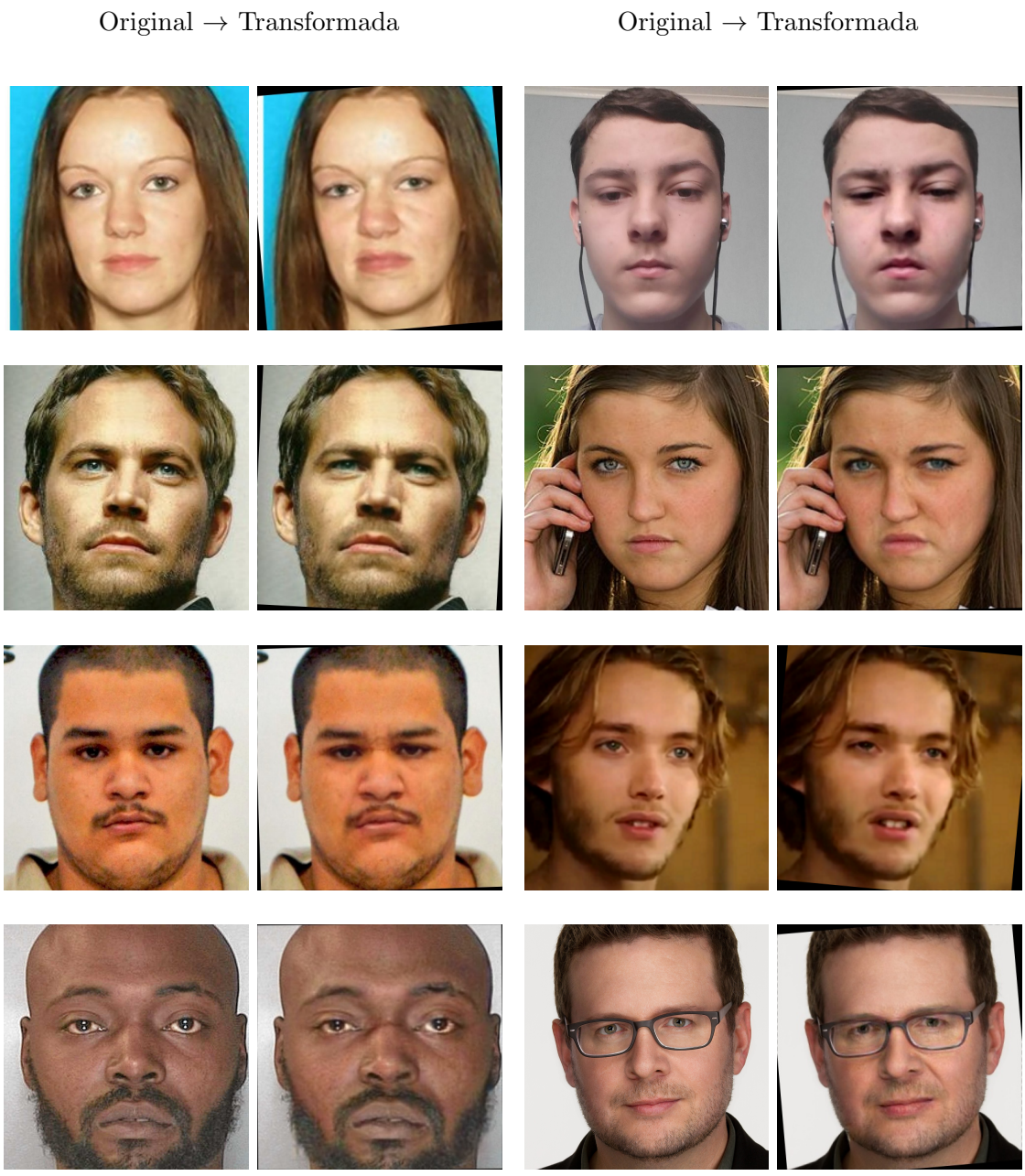


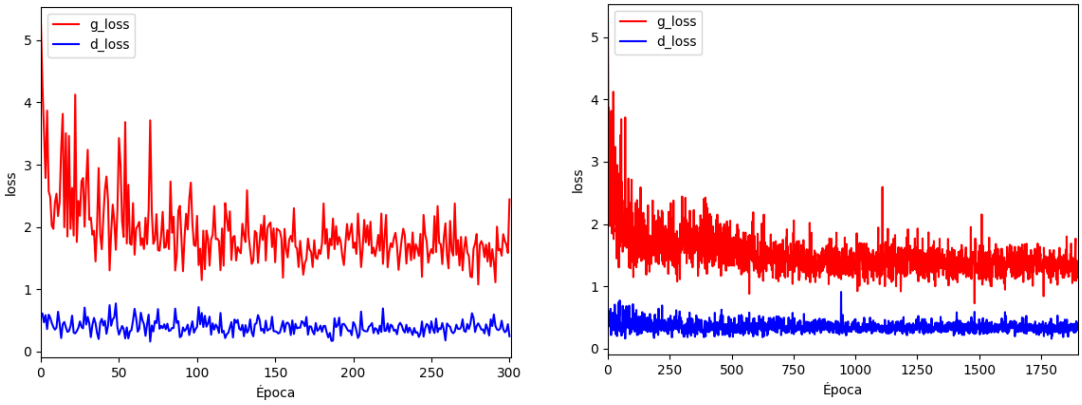
Figura 5.6.: Resultados del entrenamiento con 2000 imágenes para la emoción de asco y 200 épocas

5.5. Entrenamiento con más épocas

Con el objetivo de intentar mejorar los entrenamientos anteriores para las emociones de asco y de sorpresa, se va a realizar un entrenamiento aumentando en este caso el número de épocas que se va a utilizar. De esta manera, se van a utilizar las imágenes de la emoción de sorpresa del entrenamiento anterior y se entrenarán dos nuevos modelos: el primero durante 300 épocas y el segundo durante 2000 cuyos resultados se reflejarán en las figuras 5.8 y 5.9 respectivamente.

Por lo que respecta al nuevo entrenamiento de 300 épocas, los resultados muestran cómo apenas presenta diferencias con el equivalente de 200 y mantiene unas imágenes finales muy parecidas a las del modelo anterior. Como cambio apreciable destaca la boca del sexto sujeto, a la cual el nuevo entrenamiento le ha permitido ajustar la posición de los dientes para hacerla más natural y correcta: mientras que en el caso anterior los dientes estaban flotando, en los nuevos resultados los dientes están correctamente situados en la parte superior.

Sin embargo, los resultados del segundo entrenamiento presentan un panorama completamente diferente. Todas las imágenes finales denotan deformaciones muy marcadas que distorsionan por completo el rostro original con diferentes artefactos que alejan a los resultados de la naturalidad y credibilidad de los anteriores.



(a) 300 épocas (b) 2000 épocas

Figura 5.7.: Graficas loss

Estos resultados pueden aprovecharse para tratar de explicar las dificultades que se presentan a la hora de evaluar el estado del entrenamiento de una red GAN. En las redes neuronales tradicionales, durante el proceso de entrenamiento se dispone de los valores de *loss* y *accuracy* tanto para el conjunto de entrenamiento como para otro de test y a partir de estos valores y su evolución a través de las épocas podemos juzgar el estado del entrenamiento y marcar unos valores objetivo que sirvan para detenerlo. Por el contrario, cuando entrenamos una red GAN, el valor de precisión se pierde puesto que no tenemos unas salidas objetivo fijas con la que podamos comparar y nos permitan valorar de forma binaria si efectivamente las imágenes que estamos generando se corresponden con dichas salidas o no, por lo que solamente se dispone del *loss*.

Sin embargo, y como podemos ver en las tablas figura 5.7 y en los resultados de los respectivos entrenamientos que se han comentado anteriormente, en los valores de *loss* presentes en este caso en la CycleGAN no se ven reflejados los problemas que pueden suceder durante el entrenamiento o la evolución de los resultados que está consiguiendo. En la segunda gráfica, en algún punto a partir de las 300 épocas debería verse reflejado el momento en el que comienzan a aparecer los artefactos que deforman las imágenes finales pero no lo hace. Es por este motivo que los entrenamientos en este tipo de redes deben monitorizarse de forma visual y detener el entrenamiento en función de si dichos resultados cumplen o no las exigencias requeridas. En el caso de nuestro trabajo, de cara a conseguir los mejores resultados posibles, para cada emoción deberíamos preparar un entrenamiento extenso y comprobar cada cierto número de épocas los resultados que consigue el modelo para juzgar si detener el entrenamiento o no. Este proceso requiere de mucho tiempo puesto que el entrenamiento con 2000 épocas que se ha llevado a cabo se ha prolongado durante 3 semanas utilizando una GPU potente como es la Nvidia GTX 1080 ti y es por eso que no se ha repetido para todas las emociones.

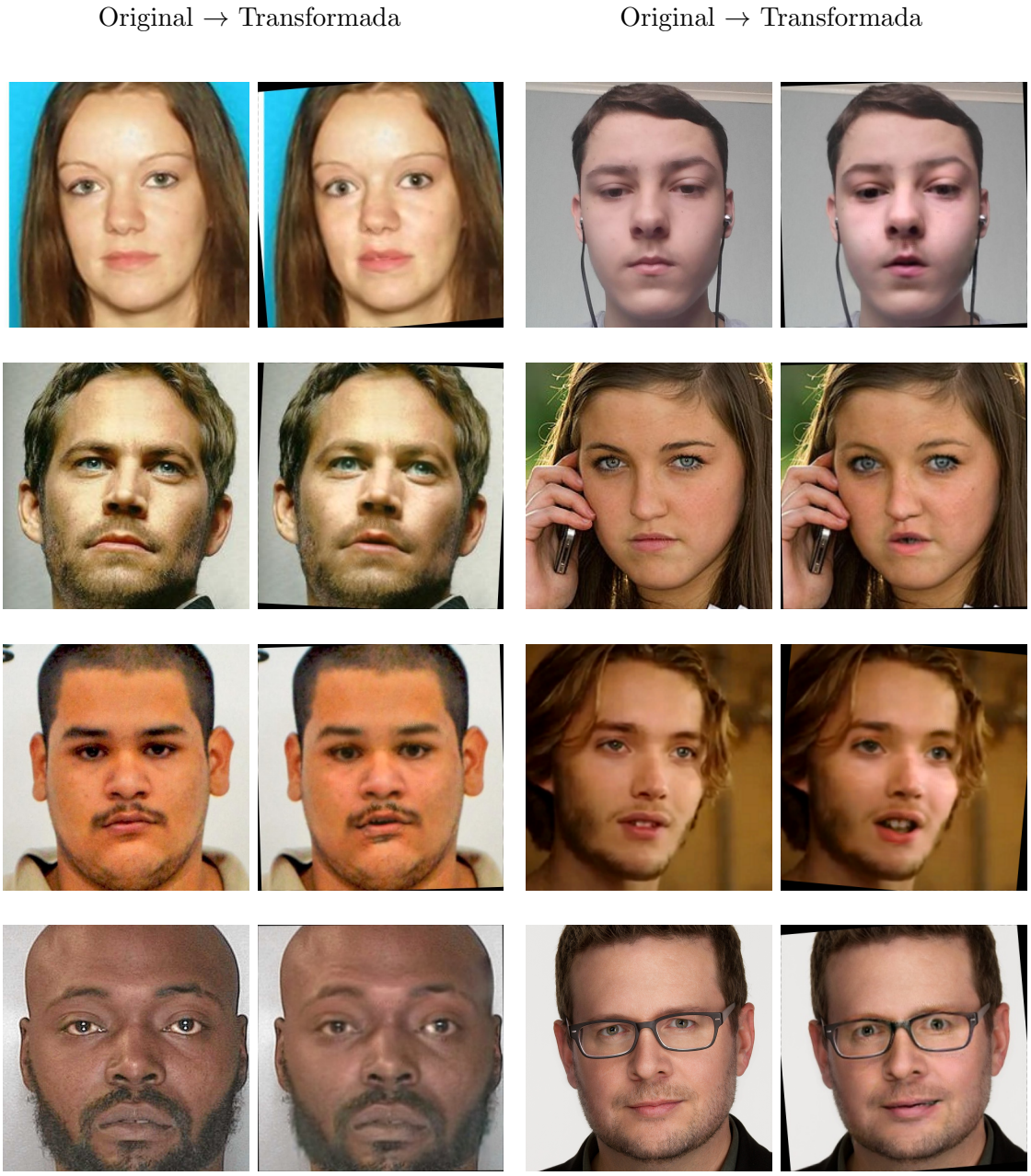


Figura 5.8.: Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 300 épocas

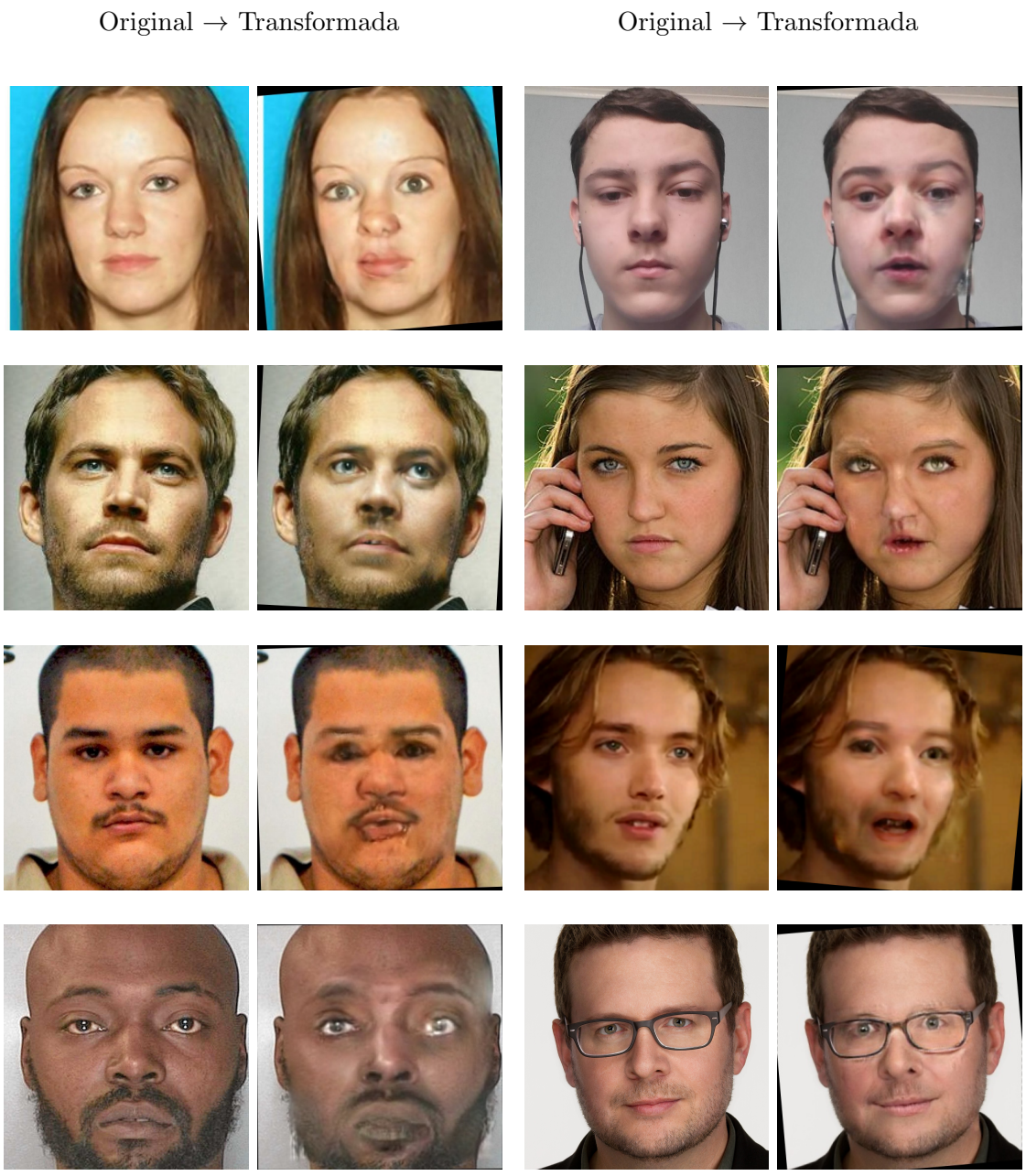


Figura 5.9.: Resultados del entrenamiento con 2000 imágenes para la emoción de sorpresa y 2000 épocas

5.6. Inferencia en tiempo real

De forma paralela a la experimentación detallada en en este capítulo, también se ha implementado un programa en Python que carga modelos entrenados y los utiliza para realizar una transformación en tiempo real de las imágenes que llegan a través de la webcam del ordenador. Debido a que para cada frame que recibe el programa procesa la imagen para detectar la cara y rotarla, pasa esta cara procesada por la red y finalmente reconstruye la imagen final, no es posible convertir todos los frames captados por la cámara y en un ordenador con las especificaciones del portátil especificadas en 3.2 se consigue una salida de unas 24 imágenes por segundo. En el siguiente enlace se puede ver el sistema en funcionamiento: `https://www.youtube.com/watch?v=JaQQVrAv9ok`

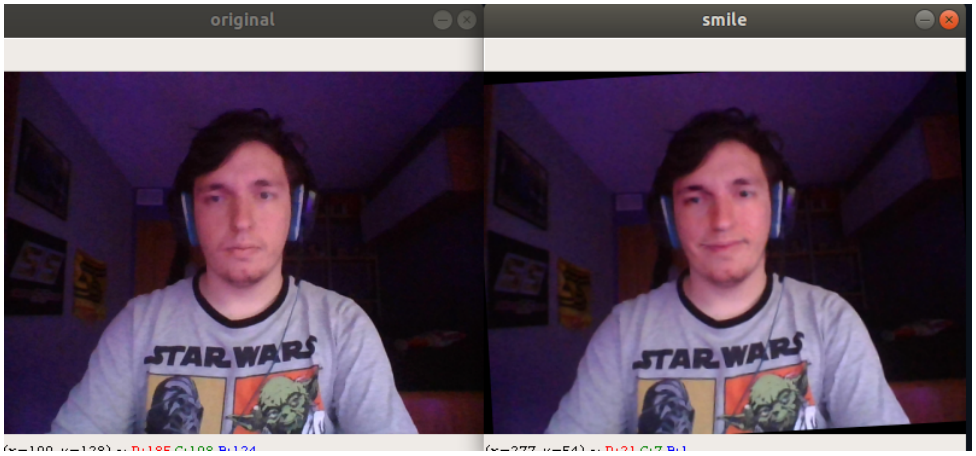


Figura 5.10.: Captura de inferencia en tiempo real a felicidad

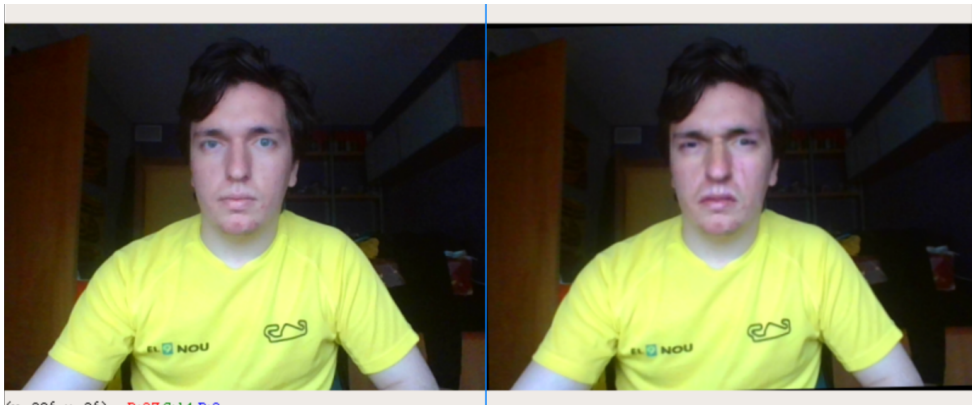


Figura 5.11.: Captura de inferencia en tiempo real a asco

6. Conclusiones

De cara a cerrar este proyecto, se va a dedicar este último capítulo a evaluar el alcance del trabajo realizado y aportar diferentes vías futuras de investigación para continuar con el proyecto y mejorar los resultados que se han conseguido.

En lo referente a los resultados que se han conseguido con este trabajo, se ha probado que se pueden utilizar redes generativas antagónicas para transformar emociones en imágenes y además, a través de la experimentación, se han demostrado formas de mejorar los resultados iniciales. De cara a conseguir estos resultados, se han cumplido los siguientes objetivos:

- Realizar un estudio sobre las redes generativas antagónicas para entender cómo funcionan y estudiar las diferentes arquitecturas para decidir cuál es la que mejor se acoplaba a nuestro problema.

- Investigar y analizar los conjuntos de datos relacionados con las expresiones faciales para finalmente elegir el que creíamos que fuese a dar mejores resultados.

- Proponer un sistema de procesado para las imágenes de nuestro dataset para mejorar los resultados del entrenamiento en la CycleGAN.

- Conseguir entrenar con éxito la red CycleGAN para transformar de rostros con emoción neutra a otra emoción diferente a través de diferentes modelos y configuraciones de entrenamiento.

Sin embargo, además de estos objetivos de carácter general del proyecto, también creo importante reflejar que en este proceso de desarrollo se ha realizado un aprendizaje paralelo de tecnologías con las que no se había trabajado anteriormente como OpenCV, dlib, Python e incluso TensorFlow y que ha implicado un aporte extra a nivel personal.

Por último me gustaría sintetizar una serie de ideas en las que se puede seguir trabajando de cara a mejorar los resultados que se han conseguido en este proyecto y otras para utilizar y evaluar los resultados en aplicaciones prácticas que, por falta de tiempo, no se han incluido en este trabajo:

- Continuar la experimentación con más configuraciones de entrenamiento para seguir mejorando los resultados finales y entrenar modelos para todas las emociones.

- Experimentar con otros conjuntos de datos y con la combinación de los mismos, desarrollando sistemas de procesado para que el conjunto resultante sea uniforme.

- Utilizar otras redes GAN para comprobar los resultados que son capaces de conseguir y para comprobar que resultados conseguiríamos con el conjunto de datos que se ha trabajado entrenando un generador que no partiera de una imagen de una persona real.

- Evaluar el impacto en la precisión y rendimiento de las arquitecturas dedicadas al reconocimiento de expresiones faciales utilizando en el entrenamiento imágenes generadas por nuestros modelos.

- Estudiar la posible aplicación de nuestros modelos en el desarrollo de una aplicación de carácter social dedicada a ayudar a las personas que tienen problemas para reconocer las emociones en otras caras.

Bibliografía

[1] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1812.04948, Dec 2018.

[2] Guim Perarnau. Icgan official repository. `https://github.com/Guim3/IcGAN`, 2015.

[3] Byoung Chul Ko. A brief review of facial emotion recognition based on visual information. *Sensors*, 18(2), 2018.

[4] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[5] Harry Pratt, Frans Coenen, Deborah M. Broadbent, Simon P. Harding, and Yalin Zheng. Convolutional neural networks for diabetic retinopathy. *Procedia Computer Science*, 90:200 – 205, 2016. 20th Conference on Medical Image Understanding and Analysis (MIUA 2016).

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1406.2661, Jun 2014.

[8] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, ab-

s/1703.10593, 2017.

[9] Jun-Yan Zhu. Cyclegan official repository. `https://github.com/junyanz/CycleGAN`, 2017.

[10] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[11] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[12] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing. *CoRR*, abs/1611.06355, 2016.

[13] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1703.05192, Mar 2017.

[14] Hui Ding, Kumar Sricharan, and Rama Chellappa. Exprgan: Facial expression editing with controllable expression intensity. *CoRR*, abs/1709.03842, 2017.

[15] Shan Li and Weihong Deng. A Deeper Look at Facial Expression Dataset Bias. *arXiv e-prints*, page arXiv:1904.11150, Apr 2019.

[16] Guoying Zhao, Xiaohua Huang, Matti Taini, Stan Z. Li, and Matti Pietikäinen. Facial expression recognition from near-infrared videos. *Image and Vision Computing*, 29(9):607 – 619, 2011.

[17] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 94–101, June 2010.

[18] Carlos Fabian Benitez-Quiroz, Ramprakash Srinivasan, Qianli Feng, Yan Wang, and Aleix M. Martínez. Emotionet challenge: Recognition of facial expressions of emotion in the wild. *CoRR*, abs/1703.01210, 2017.

[19] Ali Mollahosseini, Behzad Hassani, and Mohammad H. Mahoor. Affectnet: A database for facial expression, valence, and arousal computing in the wild. *CoRR*, abs/1708.03985, 2017.

[20] Erik Linder-Norén. Cyclegan keras implementation repository. `https://github.com/eriklindernoren/Keras-GAN/tree/master/cyclegan`, 2017.

[21] Xiaowei Hu. Cyclegan tensorflow implementation repository. `https://github.com/XHUJOY/CycleGAN-tensorflow`, 2017.

A. Anexo I: Preparación de entorno y entrenamiento de la CycleGAN

En este anexo se va a documentar el proceso necesario para preparar un entorno y comenzar a entrenar modelos de la CycleGAN con cualquier conjunto de datos utilizando la implementación en tensorflow que se ha utilizado en este trabajo.

A.1. Instalación de dependencias

De cara instalar todos los paquetes necesarios para poder entrenar la red, es recomendable utilizar un entorno virtual de Python para mantener la versión del sistema operativo lo más limpia posible. Durante el desarrollo de este proyecto se ha utilizado un entorno virtual de Python 3.6 gestionado mediante Anaconda.

Los pasos para instalar Anaconda en Ubuntu 16.04 y 18.04 han sido los siguientes[1]:

1. Descargar el instalador de Anaconda para Ubuntu desde su web oficial [2]

2. Copiar el archivo al directorio *home* y ejecutar el siguiente comando:

```
bash ./Anaconda3-5.1.0-Linux-x86_64.sh
```

3. Seguir las instrucciones del instalador y al acabar activar la instalación:

```
source ~/.bashrc
```

[1] https://medium.com/@menuram1126/how-to-install-anaconda-on-ubuntu-16-04-538009ca7936
[2] https://www.anaconda.com/distribution/#linux

Una vez instalado Anaconda, se han utilzado los siguientes comandos[3] para crear y utilizar un entorno virtual:

1. Crear el entorno virtual, en nuestro caso llamado *pythonCycle*:

```
conda update conda
conda create -n pythonCycle python=3.6 anaconda
```

2. Activar el entorno cada vez que lo vayamos a utilizar:

```
source activate pythonCycle
```

3. Instalar paquetes en el nuevo entorno virtual:

```
conda install -n pythonCycle [nombre del paquete]
```

4. Desactivar el entorno cuando lo dejamos de utilizar:

```
source deactivate
```

Cuando tenemos listo nuestro entorno virtual, podemos pasar a instalar las dependencias concretas requeridas por nuestra red listadas en el repositorio de la implementación que vamos a utilizar:

```
conda install -n pythonCycle tensorflow
conda install -n pythonCycle numpy
conda install -n pythonCycle scipy
conda install -n pythonCycle pillow
```

De cara a acelerar el proceso de entrenamiento es fuertemente recomendable utilizar la GPU. Para ello necesitaremos instalar las versiones de CUDA y cuDNN correspondientes, así como el paquete *tensorflow-gpu*[4][5]. Los pasos a seguir dependen de la distribución de

[3] https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/20/conda/
[4] https://medium.com/@taylordenouden/installing-tensorflow-gpu-on-ubuntu-18-04-89a142325138
[5] https://medium.com/@zhanwenchen/install-cuda-and-cudnn-for-tensorflow-gpu-on-ubuntu-79306e4ac04e

sistema operativo que se esté utilizando.

Una vez completados los pasos anteriores estamos en disposición de descargar la implementación y comenzar a entrenar.

A.2. Entrenamiento

El primer paso para entrenar nuestros modelos es descargar la implementación que vamos a utilizar desde su repositorio[6]. Una vez descargada, tenemos que crear una carpeta *datasets* en su directorio de tal forma que quede al mismo nivel que el archivo *main.py*. En esta carpeta crearemos otra que será la que contendrá las imágenes con las que vamos a entrenar. Podemos tener diferentes carpetas y seleccionaremos la que queramos a través de un argumento cuando vayamos a lanzar el entrenamiento.

Por ejemplo, si queremos preparar la implementación para convertir de imágenes con emoción neutra a emoción feliz, dentro de la carpeta *datasets* que hemos creado, crearemos otra con un nombre como *neutral2happy*.

Dentro de estas carpetas de cada *dataset*, tenemos que crear otras últimas 4 carpetas: *testA, testB, trainA* y *trainB*. En estas carpetas organizaremos nuestras imágenes. Es recomendable utilizar al menos 1000 imágenes de entrenamiento para cada clase, que tendríamos que poner en las carpetas de *trainA* y *trainB* respectivamente. En las carpetas de test, se recomienda poner pocas imágenes por motivo que comentaremos más adelante.

Para el ejemplo con el conjunto de datos de antes, *neutral2happy*, colocaremos las imagenes de rostros neutrales en las carpetas son el sufijo A, y las de felicidad en aquellas con el sufijo B.

Completado todo el proceso anterior, estaremos en disposición de comenzar el entrenamiento. Para entrenar un modelo utilizando la configuración predeterminada con el conjunto de datos que estamos poniendo de ejemplo, deberemos ejecutar el siguiente comando:

```
1          CUDA_VISIBLE_DEVICES=0 python3 main.py --dataset_dir=
               neutral2happy
```

[6]https://github.com/xhujoy/CycleGAN-tensorflow

CUDA_VISIBLE_DEVICES permite seleccionar la GPU que vamos a utilizar para entrenar en caso de disponer de varias.

Una vez ejecutemos el comando anterior comenzará el entrenamiento y visualizaremos en la terminal el paso y la época en la que nos encontramos.

La implementación que estamos utilizando permite acceder a los parámetros de configuración de la red a través de argumentos para *main.py*. Estos parámetros van desde el número de épocas hasta el *learning rate* a utilizar en la red. La información de todas las opciones disponibles se puede ver en el propio archivo *main.py*.

Una vez completado el entrenamiento se habrán creado diferentes directorios: *checkpoint* en el que se guardan los modelos; y *samples*, en el que durante el entrenamiento, cada cierto número de pasos se habrán guardado ejemplos de conversiones realizadas en ese momento del entrenamiento. Estas imágenes sirven para evaluar la progresión de la red a través de las épocas del entrenamiento y pueden utilizarse para valorar cuando es aconsejable detener en el entrenamiento. Las imágenes que se convierten son seleccionadas aleatoriamente de las carpetas de test y, de ser pocas, nos pr comparar la evolución de la transformación de una misma imagen, dándonos una visión más clara de dicha evolución. También se habrá creado una carpeta *logs* que puede ser utilizada junto a *tensorboard* para monitorizar los detalles del entrenamiento.

Por último, de cara a utilizar el modelo final, ejecutaremos el siguiente comando:

```
CUDA_VISIBLE_DEVICES=0 python3 main.py --dataset_dir=
   neutral2happy --phase=test --which_direction=AtoB
```

El argumento *which_direction* determina si vamos a convertir del dominio A o en este caso neutral al dominio B o feliz, o viceversa. Las imágenes que queramos convertir, tendremos que estar colocadas en las carpetas de test correspondientes. Una vez haya finalizado la ejecución, se habrá creado un directorio *test* en el que se habrán guardado los resultados junto con un archivo *html* que permite visualizar las imágenes transformadas junto a la original.